UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Resolució numèrica de les equacions de *Navier-Stokes*

Treball de Final de Grau: Annexos
Grau en Enginyeria en Vehicles Aeroespacials

Estudiant : Mario Luna López
Director: Assensi Oliva Llena
Codirector: Carles-David Pérez Segarra
Data: 10 de juny de 2018

Índex

1 Conducció de calor

```
1. //      A Two-dimensional Transient Conduction Problem by CTTC
2.
3. //      Code implemented by student Mario Luna L�pez
4.
5. #include<iostream>
6. #include<iomanip>
7. #include<math.h>
8. #include <vector>
9. using namespace std;
10.
11.
12.      //                                1. ENTRADA DE DADES
13.
14.      //1.1 Dades F�siques
15.      const double L1=0.5, L2=0.6, L3=0.4, L4=0.3, L5=0.1; //mesures en [m]
   dels materials
16.      const double rho1=1500, rho2=1600, rho3=1900, rho4=2500; //densitats
   [kg/m^3] dels materials
17.      const double cp1=750, cp2=770, cp3=810, cp4=930; //Cp [J/kgK] dels
   materials
18.      //const double lambda1=170, lambda2=140, lambda3=200, lambda4=140;
   //lambda [W/mK] dels materials
19.      const double lambda1=170, lambda2=140, lambda3=200,
   lambda4=140; //lambda [W/mK] dels materials
20.      const double Tbottom=(23+273.15), Tg=(33+273.15), alfag=9; //valors
   condicions de contorn
21.      const double qflow=60, T_0=(8+273.15); //valors condicions de contorn
   i inicials
22.
23.      //1.2 Dades num�riques
24.      const int f1=1;
25.      const int N1=10*f1, N2=20*f1; //n�mero de nodes interns en la
   direcci� x (se'n sumen 1 a extrem esquerre i dret)
26.      const int M3=30*f1, M4=10*f1; //n�mero de nodes interns en la
   direcci� y (se'n sumen 1 avall i a dalt)
27.      const double beta=0.5; //esquema num�ric (impl�cit, expl�cit,
   Crank-Nicolson)
28.      const double deltat=1;//time-step d'integraci� en el temps
29.      const double tfinal=5000; //temps final [s] resoluci� transitori
30.      const double e=10e-8;//precisi� escollida per la resoluci�
31.      const double fr=1.2; //factor de relaxaci�
32.
33.      void maxdif(double T, double Tsup, double &dif)
34.      {
35.              if(fabs(T-Tsup)>dif)
36.              {
37.                      dif=fabs(T-Tsup);
38.              }
39.      }
40.
41.
42.      int main()
43.      {
44.
45.              //                                2. C�LCULS PREVIS
46.
```

```
47.             const double deltax1=L1/N1, deltax2=L2/N2; //dist�ncia 'x'
   dels VC
48.             const double deltay1=L3/M3, deltay2=(L4+L5)/M4; //dist�ncia
   'y' dels VC
49.             //double X[N1+N2+2][M3+M4+2], Y[N1+N2+2][M3+M4+2]; //matrius
   de localitzaci� dels nodes
50.             vector <vector<double> > X(N1+N2+2,vector<double> (M3+M4+2));
51.             vector <vector<double> > Y(X);
52.             //El seg�ent 'for' s'encarrega d'emmagatzemar les
   coordenades dels nodes
53.             //Cal tenir en compte les coordenades especials com les dels
   nodes del contorn
54.             //Tamb� s�n especials els nodes separats delta/2 del contorn
55.             for(int j=0;j<(M3+M4+2);j++)
56.             {
57.                     for(int i=0;i<(N1+N2+2);i++)
58.                     {
59.                             if(j==0)
60.                             {
61.                                     Y[i][j]=0;
62.                             }
63.                             else if(j==1)
64.                             {
65.                                     Y[i][j]= deltay1/2;
66.                             }
67.                             else if(j<=M3)
68.                             {
69.                                     Y[i][j]= deltay1 + Y[i][j-1];
70.                             }
71.                             else if(j==M3+1)
72.                             {
73.                                     Y[i][j]= L3 + deltay2/2;
74.                             }
75.                             else if(j<=(M3+M4))
76.                             {
77.                                     Y[i][j]= deltay2 + Y[i][j-1];
78.                             }
79.                             else
80.                             {
81.                                     Y[i][j]=L3+L4+L5;
82.                             }
83.
84.
85.                             if(i==0)
86.                             {
87.                                     X[i][j]=0;
88.                             }
89.                             else if(i==1)
90.                             {
91.                                     X[i][j]= deltax1/2;
92.                             }
93.                             else if(i<=N1)
94.                             {
95.                                     X[i][j]= deltax1 + X[i-1][j];
96.                             }
97.                             else if(i==N1+1)
98.                             {
99.                                     X[i][j]= L1 + deltax2/2;
100.                            }
101.                            else if(i<=(N1+N2))
```

```
102.                                {
103.                                        X[i][j]= deltax2 + X[i-1][j];
104.                                }
105.                                else
106.                                {
107.                                        X[i][j]=L1+L2;
108.                                }
109.
110.                        }
111.                }
112.
113.
114.                //El seg?ents 'for' calculen les posicions verticals i
   horitzontals
115.                //De les cares dels VC
116.                double Xc[N1+N2+2],Yc[M3+M4+2];
117.
118.                for(int i=0;i<(N1+N2+1);i++)
119.                {
120.                        if(i==0) Xc[i]=0;
121.                        else if(i<=N1) Xc[i]=Xc[i-1]+deltax1;
122.                        else Xc[i]=Xc[i-1]+deltax2;
123.                }
124.                for(int j=0;j<(M3+M4+1);j++)
125.                {
126.                        if(j==0) Yc[j]=0;
127.                        else if(j<=M3) Yc[j]=Yc[j-1]+deltay1;
128.                        else Yc[j]=Yc[j-1]+deltay2;
129.                }
130.
131.
132.                //Els seg?ents 'for' calculen les dist?ncies entre nodes
133.                //Per i,j==0 la dist?ncia ?s entre node de contorn i intern
134.                //Per i=N1+N2+1 la dist?ncia ?s entre node intern i de
   contorn
135.                double deltax_vec[N1+N2+1], deltay_vec[M3+M4+1];
136.                for(int i=0;i<(N1+N2+1);i++)
137.                {
138.                        deltax_vec[i]=X[i+1][0]-X[i][0]; //El segon sub?ndex
   ?s igual quin sigui
139.                        //Hi ha la mateixa coordenada 'x' per qualsevol node
   en una vertical
140.                }
141.                for(int j=0;j<(M3+M4+1);j++)
142.                {
143.                        deltay_vec[j]=Y[0][j+1]-Y[0][j]; //El primer
   sub?ndex ?s igual quin sigui
144.                        //Hi ha la mateixa coordenada 'y' per qualsevol node
   en una horitzontal
145.                }
146.
147.
148.                //Els seg?ents 'for' calculen les superf?cies de transf.
   calor entre nodes
149.                //Sy ?s la superf?cie per transf. de nodes west i east
150.                //Sx ?s la superf?cie per transf. de nodes north i south
151.                double Sy[M3+M4], Sx[N1+N2];
152.
153.                for(int j=0;j<(M3+M4);j++)
154.                {
```

```
155.                    Sy[j]=Yc[j+1]-Yc[j];
156.            }
157.
158.            for(int i=0;i<(N1+N2);i++)
159.            {
160.                    Sx[i]=Xc[i+1]-Xc[i];
161.            }
162.
163.            //El seg?ents 'for' calculen el volum dels volums de control
   dels nodes interns
164.            //Aquest c?lcul ?s necessari pel terme transitori
165.            double V[N1+N2][M3+M4];
166.
167.            for(int j=0;j<(M3+M4);j++)
168.            {
169.                    for(int i=0;i<(N1+N2);i++)
170.                    {
171.                            V[i][j]=Sx[i]*Sy[j];
172.                    }
173.            }
174.
175.            //Els seg?ents 'for' assignen les propietats depenent del
   material
176.            //double lambda[N1+N2+2][M3+M4+2];
177.            //double rho[N1+N2+2][M3+M4+2], c[N1+N2+2][M3+M4+2];
178.            vector <vector<double> > lambda(X);
179.            vector <vector<double> > rho(X);
180.            vector <vector<double> > c(X);
181.
182.            for(int j=0;j<(M3+M4+2);j++)
183.            {
184.                    for(int i=0;i<(N1+N2+2);i++)
185.                    {
186.                            if(X[i][j]<L1&&Y[i][j]<L3)
187.                            {
188.                                    lambda[i][j]=lambda1; rho[i][j]=rho1;
    c[i][j]=cp1;
189.                            }
190.                            else if(X[i][j]<L1&&Y[i][j]>L3)
191.                            {
192.                                    lambda[i][j]=lambda3; rho[i][j]=rho3;
    c[i][j]=cp3;
193.                            }
194.                            else if(X[i][j]>L1&&Y[i][j]<(L3+L4))
195.                            {
196.                                    lambda[i][j]=lambda2; rho[i][j]=rho2;
    c[i][j]=cp2;
197.                            }
198.                            else if(X[i][j]>L1&&Y[i][j]>(L3+L4))
199.                            {
200.                                    lambda[i][j]=lambda4; rho[i][j]=rho4;
    c[i][j]=cp4;
201.                            }
202.                    }
203.            }
204.
205.
206.
207.
208.            //                              3. MAPA INICIAL
209.
```

```
210.            //double Tn[N1+N2+2][M3+M4+2]; //Temperatures als nodes per
   un temps donat
211.            //double T0[N1+N2+2][M3+M4+2]; //Temperatures als nodes a
   l'instant anterior
212.            vector <vector<double> > Tn(X);
213.            vector <vector<double> > T0(X);
214.
215.            //Els seg@ents 'for' implementen el mapa inicial de
   temperatures a la matriu T0
216.            //Tots els nodes tenen temperature T_0 excepte els de la
   isoterma Tbottom
217.            for(int j=0;j<(M3+M4+2);j++)
218.            {
219.                    for(int i=0;i<(N1+N2+2);i++)
220.                    {
221.                            if(j==0) T0[i][j]=Tbottom;
222.                            else T0[i][j]=T_0;
223.                    }
224.            }
225.
226.
227.            //La temperatura suposada ser@ igual a la de l'instant
   anterior
228.            for(int j=0;j<(M3+M4+2);j++)
229.            {
230.                    for(int i=0;i<(N1+N2+2);i++)
231.                    {
232.                            Tn[i][j]=T0[i][j];
233.                    }
234.            }
235.
236.
237.            //                              4. COEFICIENTS DE
   DISCRETITZACI@
238.
239.            //Es calculen tots els coefs. de discretitzaci@
240.            //Els termes bp dels nodes interns s@n els @nics que es
   recalcularan
241.            //Aix@ @s perqu@ depenen de la temperatura de l'instant
   anterior
242.
243.            //double ae[N1+N2+2][M3+M4+2], aw[N1+N2+2][M3+M4+2];
244.            //double ap[N1+N2+2][M3+M4+2], an[N1+N2+2][M3+M4+2];
245.            //double as[N1+N2+2][M3+M4+2], bp[N1+N2+2][M3+M4+2];
246.            //double lambdae[N1+N2+1][M3+M4+1],
   lambdaw[N1+N2+1][M3+M4+1], lambdas[N1+N2+1][M3+M4+1],
   lambdan[N1+N2+1][M3+M4+1];
247.            vector <vector<double> > ae(X),aw(X),an(X),as(X),ap(X),bp(X),
   lambdae(X),lambdaw(X),lambdas(X),lambdan(X);
248.
249.            //Nota: els nodes dels v@rtexs no tenen coeficients de
   discretitzaci@
250.            //Aquests nodes nom@s tindran una temperatura promig dels
   del voltant
251.
252.            //Els nodes del contorn inferior i dret no tenen coeficients
   de discr.
253.            //Aix@ @s perqu@ es tracta d'isotermes amb tempertatures
   conegudes
254.            for(int j=0;j<(M3+M4+2);j++)
```

```
255.             {
256.                     for(int i=0;i<(N1+N2+2);i++)
257.                     {
258.                             //S'emprar? la mitjana harm?nica (nodes
   centrats amb canvis de material)
259.
260.                             if(i==0&&j>=1&&j<=(M3+M4)) //Nodes contorn
   esquerre
261.                             {
262.                                     ae[i][j]=lambda[i+1][j]/(deltax_vec[i
   ]);
263.                                     //Pels nodes de contorn la mitjana
   harm?nica resulta lambda del node intern
264.                                     ap[i][j]=ae[i][j]+alfag;
265.                                     bp[i][j]=alfag*Tg;
266.                             }
267.
268.                             else if(j==(M3+M4+1)&&i>=1&&i<=(N1+N2)) //Nod
   es contorn superior
269.                             {
270.                                     as[i][j]=lambda[i][j-
   1]/(deltay_vec[j-1]);
271.                                     //Pels nodes de contorn la mitjana
   harm?nica resulta lambda del node intern
272.                                     ap[i][j]=as[i][j];
273.                                     bp[i][j]=qflow;
274.                             }
275.
276.                             else if(i>=1&&i<=(N1+N2)&&j>=1&&j<=(M3+M4)) /
   /Nodes interns
277.                             {
278.
279.                                     //Els ?ndexs de les superf?cies i
   volums estan desfasats (q?esti? num?rica)
280.                                     lambdae[i][j]=(X[i+1][j]-
   X[i][j])/((Xc[i]-X[i][j])/lambda[i][j]+(X[i+1][j]-Xc[i])/lambda[i+1][j]);
281.                                     lambdaw[i][j]=(X[i][j]-X[i-
   1][j])/((X[i][j]-Xc[i-1])/lambda[i][j]+(Xc[i-1]-X[i-1][j])/lambda[i-1][j]);
282.                                     lambdan[i][j]=(Y[i][j+1]-
   Y[i][j])/((Yc[j]-Y[i][j])/lambda[i][j]+(Y[i][j+1]-Yc[j])/lambda[i][j+1]);
283.                                     lambdas[i][j]=(Y[i][j]-Y[i][j-
   1])/((Y[i][j]-Yc[j-1])/lambda[i][j]+(Yc[j-1]-Y[i][j-1])/lambda[i][j-1]);
284.
285.
286.                                     ae[i][j]=beta*Sy[j-
   1]*lambdae[i][j]/deltax_vec[i];
287.                                     aw[i][j]=beta*Sy[j-
   1]*lambdaw[i][j]/deltax_vec[i-1];
288.                                     an[i][j]=beta*Sx[i-
   1]*lambdan[i][j]/deltay_vec[j];
289.                                     as[i][j]=beta*Sx[i-
   1]*lambdas[i][j]/deltay_vec[j-1];
290.
291.                                     ap[i][j]=ae[i][j]+aw[i][j]+as[i][j]+a
   n[i][j]+rho[i][j]*c[i][j]*V[i-1][j-1]/deltat;
292.
293.                                     bp[i][j]=T0[i][j]*(rho[i][j]*c[i][j]*
   V[i-1][j-1]/deltat-(1-beta)*(Sy[j-1]*lambdaw[i][j]/deltax_vec[i-1]+Sy[j-
   1]*lambdae[i][j]/deltax_vec[i]+Sx[i-1]*lambdas[i][j]/deltay_vec[j-1]+Sx[i-
   1]*lambdan[i][j]/deltay_vec[j]))+(1-beta)*(T0[i-1][j]*Sy[j-
```

```
   1]*lambdaw[i][j]/deltax_vec[i-1]+T0[i+1][j]*Sy[j-
   1]*lambdae[i][j]/deltax_vec[i]+T0[i][j-1]*Sx[i-
   1]*lambdas[i][j]/deltay_vec[j-1]+T0[i][j+1]*Sx[i-
   1]*lambdan[i][j]/deltay_vec[j]);
294.
295.                                 }
296.                         }
297.                 }
298.
299.
300.
301.                 //-----------------------------------------------------//
302.                 //--------------PROGRAMA PRINCIPAL-------------------//
303.                 //-----------------------------------------------------//
304.
305.                 for(int t=deltat;t<=tfinal;t=t+deltat) //'for' per calcular
   per cada instant de temps
306.                 {
307.                         //Es recalcula nom�s el coeficient bp ja que aquest
   dep�n de la temperatura anterior
308.                         for(int j=1;j<=(M3+M4);j++)
309.                         {
310.                                 for(int i=1;i<=(N1+N2);i++)
311.                                 {
312.                                         bp[i][j]=T0[i][j]*(rho[i][j]*c[i][j]*
   V[i-1][j-1]/deltat-(1-beta)*(Sy[j-1]*lambdaw[i][j]/deltax_vec[i-1]+Sy[j-
   1]*lambdae[i][j]/deltax_vec[i]+Sx[i-1]*lambdas[i][j]/deltay_vec[j-1]+Sx[i-
   1]*lambdan[i][j]/deltay_vec[j]))+(1-beta)*(T0[i-1][j]*Sy[j-
   1]*lambdaw[i][j]/deltax_vec[i-1]+T0[i+1][j]*Sy[j-
   1]*lambdae[i][j]/deltax_vec[i]+T0[i][j-1]*Sx[i-
   1]*lambdas[i][j]/deltay_vec[j-1]+T0[i][j+1]*Sx[i-
   1]*lambdan[i][j]/deltay_vec[j]);
313.
314.                                 }
315.
316.                         }
317.
318.
319.                         //Temperatura de la condici� de contorn dreta
320.                         for(int j=1;j<(M3+M4+1);j++)
321.                         {
322.                                 Tn[N1+N2+1][j]=T_0+t*0.005; //Ja est� en
   Kelvins
323.                         }
324.
325.
326.
327.
328.                         //                      C�LCUL ITERATIU
329.
330.                         double dif=1; //valor inicialitzat major a 'e'
   perqu� entri al 'while'
331.                         double Tsup[N1+N2+2][M3+M4+2]; //Matriu per comparar
   amb la T que es calcula
332.                         double dif_vec[N1+N2+2][M3+M4+2]; //Matriu
   difer�ncies de temperatura calculades i suposades
333.                         int it=0;
334.                         while(dif>e)
335.                         {
336.                                 dif=0; //Per perdre el valor inicialitzat
```

```
337.
338.
339.                                //S'assignen els nous valors suposats amb
   factor de relaxaci?
340.                                //Nom?s t? sentit despr?s d'haver fet la 1a
   iteraci?
341.                                if(it>0)
342.                                {
343.                                        for(int j=0;j<(M3+M4+2);j++)
344.                                        {
345.                                                for(int i=0;i<(N1+N2+2);i++)
346.                                                {
347.                                                        //Nom?s es suposa
   una nova temperatura si la difer?ncia ?s m?s gran a l'error adm?s
348.                                                        if(dif_vec[i][j]>e)
349.                                                        {
350.                                                                Tn[i][j]=Tsup
   [i][j]+fr*(Tn[i][j]-Tsup[i][j]);
351.                                                        }
352.                                                }
353.                                        }
354.                                }
355.
356.                                //Es guarda el nou mapa de temperatures
   suposat
357.
358.                                for(int j=0;j<(M3+M4+2);j++)
359.                                {
360.                                        for(int i=0;i<(N1+N2+2);i++)
361.                                        {
362.                                                Tsup[i][j]=Tn[i][j];
363.                                        }
364.                                }
365.
366.
367.                                // ESTRUCTURA DE C?LCUL PRINCIPAL
368.
369.
370.                                //C?lcul de totes les temperatures
371.                                //              M?TODE GAUSS-SEIDEL
372.                                for(int j=1;j<(M3+M4+2);j++)
373.                                {
374.                                        for(int i=0;i<(N1+N2+1);i++)
375.                                        {
376.                                                if(i>=1&&i<=(N1+N2)&&j>=1&&j<
   =(M3+M4)) //Nodes interns
377.                                                {
378.                                                        Tn[i][j]=(ae[i][j]*Tn
   [i+1][j]+aw[i][j]*Tn[i-1][j]+as[i][j]*Tn[i][j-
   1]+an[i][j]*Tn[i][j+1]+bp[i][j])/ap[i][j];
379.                                                }
380.                                                if(i==0&&j>=1&&j<=(M3+M4)) //
   Nodes contorn esquerre
381.                                                {
382.                                                        Tn[i][j]=(ae[i][j]*Tn
   [i+1][j]+bp[i][j])/ap[i][j];
383.                                                }
384.
385.                                                else if(j==(M3+M4+1)&&i>=1&&i
   <=(N1+N2)) //Nodes contorn superior
```

```
386.                                                    {
387.                                                            Tn[i][j]=(as[i][j]*Tn
    [i][j-1]+bp[i][j])/ap[i][j];
388.                                                    }
389.
390.                                            }
391.
392.                                    }
393.
394.
395.
396.
397.                                    //Els nodes dels v?rtexs s?n la mitjana dels
    del voltant
398.                                    Tn[0][0]=(Tn[1][0]+Tn[0][1])/2;
399.                                    Tn[0][M3+M4+1]=(Tn[1][M3+M4+1]+Tn[0][M3+M4])/
    2;
400.                                    Tn[N2+N1+1][M3+M4+1]=(Tn[N2+N1][M3+M4+1]+Tn[N
    2+N1+1][M3+M4])/2;
401.                                    Tn[N2+N1+1][0]=(Tn[N2+N1][0]+Tn[N2+N1+1][1])/
    2;
402.
403.                                    //Es calculen les difer?ncies i s'assigna la
    m?xima a la variable 'dif'
404.                                    for(int j=0;j<(M3+M4+2);j++)
405.                                    {
406.                                            for(int i=0;i<(N1+N2+2);i++)
407.                                            {
408.                                                    dif_vec[i][j]=fabs(Tsup[i][j]
    -Tn[i][j]);
409.                                                    maxdif(Tn[i][j], Tsup[i][j],
    dif);
410.                                            }
411.                                    }
412.
413.                            } //fi 'while'
414.
415.
416.
417.
418.                            if(t==5000)
419.                            {
420.                                    for(int j=0;j<(M3+M4+2);j++)
421.                                    {
422.                                            for(int i=0;i<(N1+N2+2);i++)
423.                                            {
424.                                                    cout<<Tn[i][j]-273.15<<" ";
425.                                            }
426.                                            cout<<endl;
427.                                    }
428.
429.                            }
430.
431.
432.                            //La temperatura nova anterior ?s la calculada (salt
    temporal)
433.                            for(int j=0;j<(M3+M4+2);j++)
434.                            {
435.                                    for(int i=0;i<(N1+N2+2);i++)
436.                                    {
437.                                            T0[i][j]=Tn[i][j];
```

```
438.                                }
439.                            }
440.                            //La nova temperatura suposada s igual a l'anterior
   calculada
441.
442.                    }
443.
444.
445.            }
```

2 Convecció-difusió

2.1 Cas 1

```
1. //      Convection-diffusion equation by CTTC - Case 1
2.
3. //      Code implemented by student Mario Luna L�pez
4.
5. #include<iostream>
6. #include<iomanip>
7. #include<math.h>
8. #include <vector>
9. using namespace std;
10.
11.
12.      //                                1. ENTRADA DE DADES
13.
14.      //1.1 Dades F�siques
15.      const double Lx=1, Ly=1; //longituds del domini
16.      const double rho=1; //densitat del fluid
17.      const double gamma=1; //coeficient de difusi�
18.      const double phi0=1, phiL=0; //valors de les condicions de contorn
19.
20.      //1.2 Dades num�riques
21.      const int N=100, M=10; //n�mero de nodes interiors en direcci� 'x' i
   'y'
22.      const double e=1e-8;//precisi� escollida per la resoluci�
23.      const double fr=1.654; //factor de relaxaci�
24.
25.
26.
27.
28.      void maxdif(double phi, double phisup, double &dif) //Validada
29.      {
30.              //Assigna el valor de la difer�ncia entre valor calculat i
   valor suposat a la variable 'dif' (si �s major que aquesta)
31.              if(fabs(phi-phisup)>dif)    dif=fabs(phi-phisup);
32.      }
33.
34.      double max(double a, double b) //Validada
35.      {
36.              //Retorna el valor m�xim entre 2 valors (comptant el signe)
37.              if(a>b) return a;
38.              else return b;
39.      }
40.
41.
42.      void malla(double Lx, double Ly, double N, double M,
   vector<vector<double> > &X, double deltax,
   vector<vector<double> > &Y, double deltay, vector<double> &Xc,
   vector<double> &Yc) //Validada
43.      {
44.              //Construeix la malla 2D rectangular del problema
45.
46.
47.              //El seg�ent 'for' s'encarreguen d'emmagatzemar les
   coordenades dels nodes
```

```
48.             //Cal tenir en compte les coordenades especials com les dels
   nodes del contorn
49.             //Tamb[?] s[?]n especials els nodes separats delta/2 del contorn
50.
51.             //Coordenades x dels nodes
52.
53.             for(int j=0;j<=(M+1);j++)
54.             {
55.                     for(int i=0;i<=(N+1);i++)
56.                     {
57.                             if(i==0) X[i][j]=0;
58.
59.                             else if(i==1) X[i][j]= deltax/2;
60.
61.                             else if(i<=N) X[i][j]= deltax + X[i-1][j];
62.
63.                             else X[i][j]=Lx;
64.
65.                     }
66.             }
67.
68.             //Coordenades y dels nodes
69.             for(int j=0;j<=(M+1);j++)
70.             {
71.                     for(int i=0;i<=(N+1);i++)
72.                     {
73.                             if(j==0) Y[i][j]=0;
74.
75.                             else if(j==1) Y[i][j]= deltay/2;
76.
77.                             else if(j<=M) Y[i][j]= deltay + Y[i][j-1];
78.
79.                             else Y[i][j]=Ly;
80.
81.                     }
82.             }
83.
84.
85.             //El seg[?]ents 'for' calculen les posicions verticals i
   horitzontals de les cares dels VC
86.             //Per un node i, li corresponen les cares 'i-1' i 'i'
87.
88.             for(int i=0;i<=N;i++)
89.             {
90.                     if(i==0) Xc[i]=0;
91.                     else Xc[i]=Xc[i-1]+deltax;
92.             }
93.             for(int j=0;j<=M;j++)
94.             {
95.                     if(j==0) Yc[j]=0;
96.                     else Yc[j]=Yc[j-1]+deltay;
97.             }
98.     }
99.
100.    double A_P(double P) //Validada
101.    {
102.            //Calcula el coeficient "Ahead" segons l'esquema introdu[?]t
103.            return max(0,pow((1-0.1*fabs(P)),5));
104.    }
105.
106.
```

```
107.     void v_horitzontal(vector<vector<double> > &u) //Validada
108.     {
109.             //Guarda el camp de velocitats de component horitzontal
110.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
111.             for(int j=0;j<=(M+1);j++)
112.             {
113.                     for(int i=0;i<=(N+1);i++)
114.                     {
115.                             u[i][j]=-10;
116.                     }
117.             }
118.     }
119.
120.     void v_vertical(vector<vector<double> > &v) //Validada
121.     {
122.             //Guarda el camp de velocitats de component vertical
123.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
124.             for(int j=0;j<=(M+1);j++)
125.             {
126.                     for(int i=0;i<=(N+1);i++)
127.                     {
128.                             v[i][j]=0;
129.                     }
130.             }
131.     }
132.
133.     void Gauss_Seidel(int N, int M, double e, double fr,
   vector<vector<double> > &phi, vector<vector<double> > &phisup,
   vector<vector<double> > ap, vector<vector<double> > ae,
   vector<vector<double> > aw, vector<vector<double> > an,
   vector<vector<double> > as, vector<vector<double> > b)
134.     {
135.             //                      C�LCUL ITERATIU
136.
137.                     double dif=1; //valor inicialitzat major a la
   precisi� escollida 'e' perqu� entri al 'while'
138.                     int it=0;
139.
140.                     while(dif>e)
141.                     {
142.
143.
144.                             dif=0; //Per perdre el valor inicialitzat
145.
146.
147.                             //S'assignen els nous valors suposats amb
   factor de relaxaci�
148.                             //Nom�s t� sentit despr�s d'haver fet la 1a
   iteraci�
149.                             if(it>0)
150.                             {
151.                                     for(int j=0;j<=(M+1);j++)
152.                                     {
153.                                             for(int i=0;i<=(N+1);i++)
154.                                             {
155.                                                     //Nom�s es suposa un
   nou valor si la difer�ncia �s m�s gran a l'error adm�s
```

```
156.                                                    if((fabs(phi[i][j]-
   phisup[i][j]))>e)
157.                                                    {
158.                                                            phi[i][j]=phi
   sup[i][j]+fr*(phi[i][j]-phisup[i][j]);
159.                                                    }
160.                                            }
161.                                    }
162.                            }
163.
164.                            //Es guarda el nou mapa de la variable phi
   suposat
165.
166.                            for(int j=0;j<=(M+1);j++)
167.                            {
168.                                    for(int i=0;i<=(N+1);i++)
169.                                    {
170.                                            phisup[i][j]=phi[i][j];
171.
172.                                    }
173.
174.                            }
175.
176.
177.                            //              ESTRUCTURA DE C?LCUL
   PRINCIPAL: M?TODE GAUSS-SEIDEL
178.
179.                            //C?lcul de tots els valors de la variable
   phi
180.
181.                            //Nodes interns
182.                            for(int j=0;j<=(M+1);j++)
183.                            {
184.                                    for(int i=1;i<=N;i++)
185.                                    {
186.                                             phi[i][j]=(ae[i][j]*phi[i+1]
   [j]+aw[i][j]*phi[i-1][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-
   1]+b[i][j])/ap[i][j]; //Condici? general
187.
188.                                    }
189.
190.                            }
191.
192.                            //Nodes superiors i inferiors
193.                            for(int i=0;i<=(N+1);i++)
194.                            {
195.                                    phi[i][0]=phi[i][1]; //Contorn
   inferior
196.                                    phi[i][M+1]=phi[i][M]; //Contorn
   superior
197.
198.                            }
199.
200.
201.
202.                            //Es calculen les difer?ncies i s'assigna la
   m?xima a la variable 'dif'
203.                            for(int j=0;j<=(M+1);j++)
204.                            {
205.                                    for(int i=0;i<=(N+1);i++)
```

```
206.                                        {
207.                                                maxdif(phi[i][j],
   phisup[i][j], dif);
208.                                        }
209.                                }
210.
211.                                it++;
212.
213.                        } //fi 'while'
214.
215.                        cout<<it<<endl;
216.
217.      }
218.
219.
220.      void TDMA(int N, vector<double> &phi, vector<double> aw,
   vector<double> ae, vector<double> b, vector<double> ap)
221.      {//La j indica l'índex constant en la línia que s'empra el TDMA en
   l'algoritme del Line-by-line
222.
223.              vector<double> P(N+2);
224.              vector<double> R(N+2);
225.
226.              //      CÀLCUL COEFICIENTS P i R
227.              for(int i=0; i<=(N+1);i++)
228.              {
229.                      if(i==0){ P[i]=ae[i]/ap[i]; R[i]=b[i]/ap[i];}
230.                      else if(i==(N+1)){ P[i]=0; R[i]=(b[i]+aw[i]*R[i-
   1])/(ap[i]-aw[i]*P[i-1]);}
231.                      else{ P[i]=ae[i]/(ap[i]-aw[i]*P[i-
   1]); R[i]=(b[i]+aw[i]*R[i-1])/(ap[i]-aw[i]*P[i-1]);}
232.              }
233.
234.              //      CÀLCUL VARIABLE PHI
235.              for(int i=(N+1); i>=0;i--)
236.              {
237.                      if(i==(N+1)) phi[i]=R[i];
238.                      else phi[i]=P[i]*phi[i+1]+R[i];
239.              }
240.      }
241.
242.
243.      void line_by_line_solver(int N, int M, double e,
   vector<vector<double> > &phi, vector<vector<double> > aw,
   vector<vector<double> > ae, vector<vector<double> > as,
   vector<vector<double> > an, vector<vector<double> > b,
   vector<vector<double> > ap)
244.      {
245.
246.
247.              vector<double> b_line(N+2), phi_line(N+2), ae_line(N+2),
   aw_line(N+2), ap_line(N+2); //1r escombrat
248.              vector<double> b_line_(M+2), phi_line_(M+2), as_line(M+2),
   an_line(M+2), ap_line_(M+2); //2n escombrat
249.              vector<vector<double> > phisup(phi);
250.              double dif=1;//valor inicialitzat major a la precisió
   escollida 'e' perquè entri al 'while'
251.              int it=0;
252.
253.              while(dif>e)
```

```
254.             {
255.                     dif=0;//Per perdre el valor inicialitzat
256.
257.                     //Es guarda el nou mapa de la variable phi
   suposat
258.                     for(int j=0;j<=(M+1);j++)
259.                     {
260.                             for(int i=0;i<=(N+1);i++)
261.                             {
262.                                     phisup[i][j]=phi[i][j];
263.                             }
264.                     }
265.
266.
267.
268.                     //              1r ESCOMBRAT: Escombrat de baix a
   dalt
269.                     for(int j=0; j<=(M+1);j++)
270.                     {
271.                             // 1. Es guarden els valors de la l�nia
272.                             for(int i=0; i<=(N+1);i++)
273.                             {
274.                                     phi_line[i]=phi[i][j];  ae_line[i]=ae
   [i][j];    aw_line[i]=aw[i][j];    ap_line[i]=ap[i][j];
275.                             }
276.
277.                             //      2. Es recalcula la b_line
278.                             if(j==0) //No nodes sud
279.                             {
280.                                     for(int i=0; i<=(N+1);i++) {b_line[i]
   =b[i][j]+an[i][j]*phi[i][j+1];}
281.                             }
282.                             else if(j==(M+1)) //No nodes nord
283.                             {
284.                                     for(int i=0; i<=(N+1);i++){b_line[i]=
   b[i][j]+as[i][j]*phi[i][j-1];}
285.                             }
286.                             else
287.                             {
288.                                     for(int i=0; i<=(N+1);i++){b_line[i]=
   b[i][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-1];}
289.                             }
290.
291.                             //      3. TDMA SOLVER
292.                             TDMA(N,phi_line,aw_line,ae_line,b_line,ap_lin
   e);
293.
294.
295.                             // 4. Es guarden els valors calculats
296.                             for(int i=0; i<=(N+1);i++){phi[i][j]=phi_line
   [i];}
297.                     }
298.
299.
300.
301.                     //              2n ESCOMBRAT:Escombrat d'esquerra a
   dreta
302.                     for(int i=0; i<=(N+1);i++)
303.                     {
304.                             // 1. Es guarden els valors de la l�nia
305.                             for(int j=0; j<=(M+1);j++)
```

```
306.                                {
307.                                        phi_line_[j]=phi[i][j]; as_line[j]=as
   [i][j];    an_line[j]=an[i][j];    ap_line_[j]=ap[i][j];
308.                                }
309.
310.                                //      2. Es recalcula la b_line
311.                                if(i==0) //No nodes oest
312.                                {
313.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+ae[i][j]*phi[i+1][j];}
314.                                }
315.                                else if(i==(N+1)) //No nodes est
316.                                {
317.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+aw[i][j]*phi[i-1][j];}
318.                                }
319.                                else
320.                                {
321.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+ae[i][j]*phi[i+1][j]+aw[i][j]*phi[i-1][j];}
322.                                }
323.
324.                                //      3. TDMA SOLVER
325.                                TDMA(M,phi_line_,as_line,an_line,b_line_,ap_l
   ine_);
326.
327.                                // 4. Es guarden els valors calculats
328.                                for(int j=0; j<=(M+1);j++){phi[i][j]=phi_line
   [j];}
329.                        }
330.
331.
332.                        //Es calculen les difer�ncies i s'assigna la m�xima
   a la variable 'dif'
333.                                for(int j=0;j<=(M+1);j++)
334.                                {
335.                                        for(int i=0;i<=(N+1);i++)
336.                                        {
337.                                                maxdif(phi[i][j],
   phisup[i][j], dif);
338.                                        }
339.                                        //cout<<j<<" ";cout<<endl;
340.                                }
341.                        it++;
342.
343.                }//Fi while
344.
345.
346.        }
347.
348.        int main()
349.        {
350.
351.                //                              2. C�LCULS PREVIS
352.
353.                const double deltax=Lx/N, deltay=Ly/M; //dist�ncia 'x' i 'y'
   dels VC
354.                vector <vector<double> > X(N+2,vector<double> (M+2)); //vecto
   rs de localitzaci� dels nodes
355.                vector <vector<double> > Y(X);
356.                vector<double>  Xc(N+1), Yc(M+1);
```

```
357.
358.
359.            //Creaci? de la malla
360.            malla(Lx, Ly, N, M, X, deltax, Y, deltay, Xc, Yc);
361.
362.
363.            vector <vector<double> > u(X), v(X); //vectors de les
   components horitz. i vert. de la velocitat
364.
365.            //Introducci? del camp de velocitats
366.            v_horitzontal(u);
367.            v_vertical(v);
368.
369.
370.            //                              3. COEFICIENTS DE
   DISCRETITZACI?
371.
372.            //Es calculen tots els coefs. de discretitzaci?
373.            vector <vector<double> > ae(X),aw(X),an(X),as(X),ap(X),b(X);
374.            double De,Dw,Dn,Ds,Fe,Fw,Fn,Fs,Pe,Pw,Pn,Ps;
375.
376.            for(int j=0;j<=(M+1);j++)
377.            {
378.                    for(int i=0;i<=(N+1);i++)
379.                    {
380.                            b[i][j]=0;
381.                    }
382.            }
383.
384.            //Els nodes dels v?rtexs no tenen coeficients de discr.
   (tindran un valor phi promig dels del voltant)
385.
386.            for(int j=1;j<=M;j++)
387.            {
388.                    for(int i=1;i<=N;i++)
389.                    {
390.                            //Cara est
391.                            De=gamma*deltay/fabs(X[i+1][j]-X[i][j]);
392.                            Fe=rho*u[i][j]*deltay;
393.                            Pe=Fe/De;
394.
395.                            ae[i][j]=De*A_P(Pe)+max(-Fe,0);
396.
397.
398.                            //Cara oest
399.                            Dw=gamma*deltay/fabs(X[i][j]-X[i-1][j]);
400.                            Fw=rho*u[i][j]*deltay;
401.                            Pw=Fw/Dw;
402.
403.                            aw[i][j]=Dw*A_P(Pw)+max(Fw,0);
404.
405.                            //Cara nord
406.                            Dn=gamma*deltax/fabs(Y[i][j+1]-Y[i][j]);
407.                            Fn=rho*v[i][j]*deltax;
408.                            Pn=Fn/Dn;
409.
410.                            an[i][j]=Dn*A_P(Pn)+max(-Fn,0);
411.
412.                            //Cara sud
413.                            Ds=gamma*deltax/fabs(Y[i][j]-Y[i][j-1]);
```

```
414.                                Fs=rho*v[i][j]*deltax;
415.                                Ps=Fs/Ds;
416.
417.                                as[i][j]=Ds*A_P(Ps)+max(Fs,0);
418.
419.                                //Node P
420.                                ap[i][j]=ae[i][j]+aw[i][j]+an[i][j]+as[i][j];
421.
422.                                //Terme independent
423.                                //�s nul si no hi ha terme font i no �s
   transitori
424.
425.                        }
426.                }
427.
428.
429.
430.                for(int i=1;i<=N;i++)
431.                {
432.                        ap[i][M+1]=1; ae[i][M+1]=0; aw[i][M+1]=0; an[i][M+1]=
   0; as[i][M+1]=1; b[i][M+1]=0; //Nodes superiors
433.                        ap[i][0]=1; ae[i][0]=0; aw[i][0]=0; an[i][0]=1; as[i]
   [0]=0; b[i][0]=0; //Nodes inferiors
434.                }
435.
436.
437.                //                                4. PROGRAMA PRINCIPAL
438.
439.                vector <vector<double> > phi(X),phisup(X);
440.
441.                //S'aplicaran les condicions de contorn de Dirichlet
442.
443.
444.                for(int j=1;j<=M;j++)
445.                {
446.                        phi[0][j]=phi0; //Condici� esquerra
447.                        phi[N+1][j]=phiL; //Condici� dreta
448.                }
449.
450.
451.                //                        4.1 Valors suposats inicialment
452.
453.                //Se suposa la zona esquerra amb el valor de la CC esquerra
454.                for(int j=1;j<=M;j++)
455.                {
456.                        for(int i=1;i<=(N/2);i++)
457.                        {
458.                                phi[i][j]=phiL;
459.                        }
460.                }
461.
462.                //Se suposa la zona dreta amb el valor de la CC dreta
463.                for(int j=1;j<=M;j++)
464.                {
465.                        for(int i=(N/2+1);i<=N;i++)
466.                        {
467.                                phi[i][j]=phiL;
468.                        }
469.                }
470.
471.                //S'aplica la CC de Neumann
```

```
472.
473.             for(int i=0;i<=(N+1);i++)
474.             {
475.                     phi[i][0]=phi[i][1]; //Contorn inferior
476.                     phi[i][M+1]=phi[i][M]; //Contorn superior
477.             }
478.
479.
480.
481.             //                       4.2 SOLVER: GAUSS-SEIDEL
482.             Gauss_Seidel(N, M, e, fr, phi, phisup, ap, ae, aw, an, as,
    b);
483.
484.             //line_by_line_solver(N, M, e, phi, aw, ae, as, an, b, ap);
485.
486.
487.             for(int j=0;j<=(M+1);j++)
488.             {
489.                     for(int i=0;i<=(N+1);i++)
490.                     {
491.                             cout<<phi[i][j]<<" ";
492.                     }
493.
494.                     cout<<endl;
495.             }
496.
497.     }
```

2.2 Cas 2

```
1. //      Convection-diffusion equation by CTTC - Case 2
2.
3. //      Code implemented by student Mario Luna L�pez
4.
5. #include<iostream>
6. #include<iomanip>
7. #include<math.h>
8. #include <vector>
9. using namespace std;
10.
11.
12.      //                                1. ENTRADA DE DADES
13.
14.      //1.1 Dades F�siques
15.      const double Lx=1, Ly=1; //longituds del domini
16.      const double rho=1; //densitat del fluid
17.      const double gamma=1; //coeficient de difusi�
18.      const double phi0=1, phiL=0; //valors de les condicions de contorn
19.
20.      //1.2 Dades num�riques
21.      const int N=100, M=1; //n�mero de nodes interiors en direcci� 'x' i
   'y'
22.      const double e=1e-8;//precisi� escollida per la resoluci�
23.      const double fr=1.654; //factor de relaxaci�
24.
25.
26.
27.
28.      void maxdif(double phi, double phisup, double &dif)
29.      {
30.              //Assigna el valor de la difer�ncia entre valor calculat i
   valor suposat a la variable 'dif' (si �s major que aquesta)
31.              if(fabs(phi-phisup)>dif)    dif=fabs(phi-phisup);
32.      }
33.
34.      double max(double a, double b)
35.      {
36.              //Retorna el valor m�xim entre 2 valors (comptant el signe)
37.              if(a>b) return a;
38.              else return b;
39.      }
40.
41.
42.      void malla(double Lx, double Ly, double N, double M,
   vector<vector<double> > &X, double deltax,
   vector<vector<double> > &Y, double deltay, vector<double> &Xc,
   vector<double> &Yc) //Validada
43.      {
44.              //Construeix la malla 2D rectangular del problema
45.
46.
47.              //El seg�ent 'for' s'encarreguen d'emmagatzemar les
   coordenades dels nodes
48.              //Cal tenir en compte les coordenades especials com les dels
   nodes del contorn
```

```
49.             //Tamb□ s□n especials els nodes separats delta/2 del contorn
50.
51.             //Coordenades x dels nodes
52.
53.             for(int j=0;j<=(M+1);j++)
54.             {
55.                     for(int i=0;i<=(N+1);i++)
56.                     {
57.                             if(i==0) X[i][j]=0;
58.
59.                             else if(i==1) X[i][j]= deltax/2;
60.
61.                             else if(i<=N) X[i][j]= deltax + X[i-1][j];
62.
63.                             else X[i][j]=Lx;
64.
65.                     }
66.             }
67.
68.             //Coordenades y dels nodes
69.             for(int j=0;j<=(M+1);j++)
70.             {
71.                     for(int i=0;i<=(N+1);i++)
72.                     {
73.                             if(j==0) Y[i][j]=0;
74.
75.                             else if(j==1) Y[i][j]= deltay/2;
76.
77.                             else if(j<=M) Y[i][j]= deltay + Y[i][j-1];
78.
79.                             else Y[i][j]=Ly;
80.
81.                     }
82.             }
83.
84.
85.             //El seg□ents 'for' calculen les posicions verticals i
    horitzontals de les cares dels VC
86.             //Per un node i, li corresponen les cares 'i-1' i 'i'
87.
88.             for(int i=0;i<=N;i++)
89.             {
90.                     if(i==0) Xc[i]=0;
91.                     else Xc[i]=Xc[i-1]+deltax;
92.             }
93.             for(int j=0;j<=M;j++)
94.             {
95.                     if(j==0) Yc[j]=0;
96.                     else Yc[j]=Yc[j-1]+deltay;
97.             }
98.     }
99.
100.    double A_P(double P) //Validada
101.    {
102.            //Calcula el coeficient "Ahead" segons l'esquema introdu□t
103.            return max(0,pow((1-0.1*fabs(P)),5));
104.    }
105.
106.
107.    void v_horitzontal(vector<vector<double> > &u) //Validada
108.    {
```

```
109.             //Guarda el camp de velocitats de component horitzontal
110.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
111.             for(int j=0;j<=(M+1);j++)
112.             {
113.                     for(int i=0;i<=(N+1);i++)
114.                     {
115.                             u[i][j]=0;
116.                     }
117.             }
118.     }
119.
120.     void v_vertical(vector<vector<double> > &v) //Validada
121.     {
122.             //Guarda el camp de velocitats de component vertical
123.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
124.             for(int j=0;j<=(M+1);j++)
125.             {
126.                     for(int i=0;i<=(N+1);i++)
127.                     {
128.                             v[i][j]=-1;
129.                     }
130.             }
131.     }
132.
133.     void Gauss_Seidel(int N, int M, double e, double fr,
   vector<vector<double> > &phi, vector<vector<double> > &phisup,
   vector<vector<double> > ap, vector<vector<double> > ae,
   vector<vector<double> > aw, vector<vector<double> > an,
   vector<vector<double> > as, vector<vector<double> > b)
134.     {
135.             //                      C�LCUL ITERATIU
136.
137.                     double dif=1; //valor inicialitzat major a la
   precisi� escollida 'e' perqu� entri al 'while'
138.                     int it=0;
139.
140.                     while(dif>e)
141.                     {
142.
143.                             dif=0; //Per perdre el valor inicialitzat
144.
145.
146.                             //S'assignen els nous valors suposats amb
   factor de relaxaci�
147.                             //Nom�s t� sentit despr�s d'haver fet la 1a
   iteraci�
148.                             if(it>0)
149.                             {
150.                                     for(int j=0;j<=(M+1);j++)
151.                                     {
152.                                             for(int i=0;i<=(N+1);i++)
153.                                             {
154.                                                     //Nom�s es suposa un
   nou valor si la difer�ncia �s m�s gran a l'error adm�s
155.                                                     if((fabs(phi[i][j]-
   phisup[i][j]))>e)
156.                                                     {
```

```
157.                                                                    phi[i][j]=phi
   sup[i][j]+fr*(phi[i][j]-phisup[i][j]);
158.                                                            }
159.                                                    }
160.                                            }
161.                                    }
162.
163.                                    //Es guarda el nou mapa de la variable phi
   suposat
164.
165.                                    for(int j=0;j<=(M+1);j++)
166.                                    {
167.                                            for(int i=0;i<=(N+1);i++)
168.                                            {
169.                                                    phisup[i][j]=phi[i][j];
170.
171.                                            }
172.
173.                                    }
174.
175.
176.                                    //              ESTRUCTURA DE C�LCUL
   PRINCIPAL: M�TODE GAUSS-SEIDEL
177.
178.                                    //C�lcul de tots els valors de la variable
   phi
179.
180.                                    //Nodes interns
181.                                    for(int j=0;j<=(M+1);j++)
182.                                    {
183.                                            for(int i=1;i<=N;i++)
184.                                            {
185.                                                     phi[i][j]=(ae[i][j]*phi[i+1]
   [j]+aw[i][j]*phi[i-1][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-
   1]+b[i][j])/ap[i][j]; //Condici� general
186.
187.                                            }
188.
189.                                    }
190.
191.                                    //Nodes superiors i inferiors
192.                                    for(int i=0;i<=(N+1);i++)
193.                                    {
194.                                            phi[i][0]=phi[i][1]; //Contorn
   inferior
195.                                            phi[i][M+1]=phi[i][M]; //Contorn
   superior
196.
197.                                    }
198.
199.
200.
201.                                    //Es calculen les difer�ncies i s'assigna la
   m�xima a la variable 'dif'
202.                                    for(int j=0;j<=(M+1);j++)
203.                                    {
204.                                            for(int i=0;i<=(N+1);i++)
205.                                            {
206.                                                    maxdif(phi[i][j],
   phisup[i][j], dif);
```

```
207.                                    }
208.                            }
209.
210.                            it++;
211.                    } //fi 'while'
212.
213.                    cout<<it<<endl;
214.
215.    }
216.
217.
218.    int main()
219.    {
220.
221.            //                              2. C�LCULS PREVIS
222.
223.            const double deltax=Lx/N, deltay=Ly/M; //dist�ncia 'x' i 'y'
   dels VC
224.            vector <vector<double> > X(N+2,vector<double> (M+2)); //vecto
   rs de localitzaci� dels nodes
225.            vector <vector<double> > Y(X);
226.            vector<double>  Xc(N+1), Yc(M+1);
227.
228.
229.            //Creaci� de la malla
230.            malla(Lx, Ly, N, M, X, deltax, Y, deltay, Xc, Yc);
231.
232.
233.            vector <vector<double> > u(X), v(X); //vectors de les
   components horitz. i vert. de la velocitat
234.
235.            //Introducci� del camp de velocitats
236.            v_horitzontal(u);
237.            v_vertical(v);
238.
239.
240.            //                              3. COEFICIENTS DE
   DISCRETITZACI�
241.
242.            //Es calculen tots els coefs. de discretitzaci�
243.            vector <vector<double> > ae(X),aw(X),an(X),as(X),ap(X),b(X);
244.            double De,Dw,Dn,Ds,Fe,Fw,Fn,Fs,Pe,Pw,Pn,Ps;
245.
246.            for(int j=0;j<=(M+1);j++)
247.            {
248.                    for(int i=0;i<=(N+1);i++)
249.                    {
250.                            b[i][j]=0;
251.                    }
252.            }
253.
254.            //Els nodes dels v�rtexs no tenen coeficients de discr.
   (tindran un valor phi promig dels del voltant)
255.
256.            for(int j=1;j<=M;j++)
257.            {
258.                    for(int i=1;i<=N;i++)
259.                    {
260.                            //Cara est
261.                            De=gamma*deltay/fabs(X[i+1][j]-X[i][j]);
```

```
262.                                Fe=rho*u[i][j]*deltay;
263.                                Pe=Fe/De;
264.
265.                                ae[i][j]=De*A_P(Pe)+max(-Fe,0);
266.
267.
268.                                //Cara oest
269.                                Dw=gamma*deltay/fabs(X[i][j]-X[i-1][j]);
270.                                Fw=rho*u[i][j]*deltay;
271.                                Pw=Fw/Dw;
272.
273.                                aw[i][j]=Dw*A_P(Pw)+max(Fw,0);
274.
275.                                //Cara nord
276.                                Dn=gamma*deltax/fabs(Y[i][j+1]-Y[i][j]);
277.                                Fn=rho*v[i][j]*deltax;
278.                                Pn=Fn/Dn;
279.
280.                                an[i][j]=Dn*A_P(Pn)+max(-Fn,0);
281.
282.                                //Cara sud
283.                                Ds=gamma*deltax/fabs(Y[i][j]-Y[i][j-1]);
284.                                Fs=rho*v[i][j]*deltax;
285.                                Ps=Fs/Ds;
286.
287.                                as[i][j]=Ds*A_P(Ps)+max(Fs,0);
288.
289.                                //Node P
290.                                ap[i][j]=ae[i][j]+aw[i][j]+an[i][j]+as[i][j];
291.
292.                                //Terme independent
293.                                //�s nul si no hi ha terme font i no �s
    transitori
294.
295.                        }
296.
297.                }
298.
299.
300.
301.
302.                for(int i=1;i<=N;i++) //OK
303.                {
304.                        ap[i][M+1]=1; ae[i][M+1]=0; aw[i][M+1]=0; an[i][M+1]=
    0; as[i][M+1]=1; b[i][M+1]=0; //Nodes superiors
305.                        ap[i][0]=1; ae[i][0]=0; aw[i][0]=0; an[i][0]=1; as[i]
    [0]=0; b[i][0]=0; //Nodes inferiors
306.                }
307.
308.
309.                //                                4. PROGRAMA PRINCIPAL
310.
311.                vector <vector<double> > phi(X),phisup(X);
312.
313.                //S'aplicaran les condicions de contorn de Dirichlet
314.
315.
316.                for(int j=1;j<=M;j++) //OK
317.                {
318.                        phi[0][j]=phi0; //Condici� esquerra
319.                        phi[N+1][j]=phiL; //Condici� dreta
```

```
320.             }
321.
322.
323.
324.             //                      4.1 Valors suposats inicialment
325.
326.             //Se suposa la zona esquerra amb el valor de la CC esquerra
327.             for(int j=1;j<=M;j++)
328.             {
329.                     for(int i=1;i<=(N/2);i++)
330.                     {
331.                             phi[i][j]=phiL;
332.                     }
333.             }
334.
335.             //Se suposa la zona dreta amb el valor de la CC dreta
336.             for(int j=1;j<=M;j++)
337.             {
338.                     for(int i=(N/2+1);i<=N;i++)
339.                     {
340.                             phi[i][j]=phiL;
341.                     }
342.             }
343.
344.             //S'aplica la CC de Neumann
345.
346.             for(int i=0;i<=(N+1);i++)
347.             {
348.                     phi[i][0]=phi[i][1]; //Contorn inferior
349.                     phi[i][M+1]=phi[i][M]; //Contorn superior
350.             }
351.
352.
353.
354.             //                      4.2 SOLVER: GAUSS-SEIDEL
355.             Gauss_Seidel(N, M, e, fr, phi, phisup, ap, ae, aw, an, as,
    b);
356.
357.             for(int j=0;j<=(M+1);j++)
358.             {
359.                     for(int i=0;i<=(N+1);i++)
360.                     {
361.                             cout<<phi[i][j]<<" ";
362.                     }
363.
364.                     cout<<endl;
365.             }
366.
367.     }
```

2.3 Cas 3

```
1. //      Convection-diffusion equation by CTTC - Case 3
2.
3. //      Code implemented by student Mario Luna L�pez
4.
5. #include<iostream>
6. #include<iomanip>
7. #include<math.h>
8. #include <vector>
9. #include <fstream>
10.      using namespace std;
11.
12.
13.      //                                1. ENTRADA DE DADES
14.
15.      //1.1 Dades F�siques
16.      const double Lx=1, Ly=1; //longituds del domini
17.      const double rho=1; //densitat del fluid
18.      const double gamma=1; //coeficient de difusi�
19.      const double phi1=1, phi2=0; //valors de les condicions de contorn
20.      const double vel=1e12; //m�dul velocitat
21.      const double angle=45; //angle de la velocitat (en graus)
22.      const double pi=acos(-1); //constant pi
23.
24.      //1.2 Dades num�riques
25.      const int N=1000, M=1000; //n�mero de nodes interiors en direcci�
   'x' i 'y'
26.      const double e=1e-8;//precisi� escollida per la resoluci�
27.      const double fr=1.2; //factor de relaxaci�
28.
29.
30.
31.
32.      void maxdif(double phi, double phisup, double &dif) //Validada
33.      {
34.              //Assigna el valor de la difer�ncia entre valor calculat i
   valor suposat a la variable 'dif' (si �s major que aquesta)
35.              if(fabs(phi-phisup)>dif)    dif=fabs(phi-phisup);
36.      }
37.
38.      double max(double a, double b) //Validada
39.      {
40.              //Retorna el valor m�xim entre 2 valors (comptant el signe)
41.              if(a>b) return a;
42.              else return b;
43.      }
44.
45.
46.      void malla(double Lx, double Ly, double N, double M,
   vector<vector<double> > &X, double deltax,
   vector<vector<double> > &Y, double deltay, vector<double> &Xc,
   vector<double> &Yc) //Validada
47.      {
48.              //Construeix la malla 2D rectangular del problema
49.
50.
```

```
51.             //El seg?ent 'for' s'encarreguen d'emmagatzemar les
   coordenades dels nodes
52.             //Cal tenir en compte les coordenades especials com les dels
   nodes del contorn
53.             //Tamb? s?n especials els nodes separats delta/2 del contorn
54.
55.             //Coordenades x dels nodes
56.
57.             for(int j=0;j<=(M+1);j++)
58.             {
59.                     for(int i=0;i<=(N+1);i++)
60.                     {
61.                             if(i==0) X[i][j]=0;
62.
63.                             else if(i==1) X[i][j]= deltax/2;
64.
65.                             else if(i<=N) X[i][j]= deltax + X[i-1][j];
66.
67.                             else X[i][j]=Lx;
68.
69.                     }
70.             }
71.
72.             //Coordenades y dels nodes
73.             for(int j=0;j<=(M+1);j++)
74.             {
75.                     for(int i=0;i<=(N+1);i++)
76.                     {
77.                             if(j==0) Y[i][j]=0;
78.
79.                             else if(j==1) Y[i][j]= deltay/2;
80.
81.                             else if(j<=M) Y[i][j]= deltay + Y[i][j-1];
82.
83.                             else Y[i][j]=Ly;
84.
85.                     }
86.             }
87.
88.
89.             //El seg?ents 'for' calculen les posicions verticals i
   horitzontals de les cares dels VC
90.             //Per un node i, li corresponen les cares 'i-1' i 'i'
91.
92.             for(int i=0;i<=N;i++)
93.             {
94.                     if(i==0) Xc[i]=0;
95.                     else Xc[i]=Xc[i-1]+deltax;
96.             }
97.             for(int j=0;j<=M;j++)
98.             {
99.                     if(j==0) Yc[j]=0;
100.                    else Yc[j]=Yc[j-1]+deltay;
101.            }
102.    }
103.
104.    double A_P(double P) //Validada
105.    {
106.            //Calcula el coeficient "Ahead" segons l'esquema introdu?t
107.            return max(0,pow((1-0.1*fabs(P)),5));
108.    }
```

```
109.
110.
111.     void v_horitzontal(vector<vector<double> > &u, double vel, double ang
   le)
112.     {
113.             //Guarda el camp de velocitats de component horitzontal
114.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
115.             for(int j=0;j<=(M+1);j++)
116.             {
117.                     for(int i=0;i<=(N+1);i++)
118.                     {
119.                             u[i][j]=vel*cos(angle*pi/180); //Cal passar
   el valor a radians
120.                     }
121.             }
122.     }
123.
124.     void v_vertical(vector<vector<double> > &v, double vel, double angle)
125.     {
126.             //Guarda el camp de velocitats de component vertical
127.             //LES VELOCITATS CAL QUE SIGUIN LES CORRESPONENTS AL
   TRAVESSAR LA CARA DEL VC (NO del node P)
128.             for(int j=0;j<=(M+1);j++)
129.             {
130.                     for(int i=0;i<=(N+1);i++)
131.                     {
132.                             v[i][j]=vel*sin(angle*pi/180); //Cal passar
   el valor a radians
133.                     }
134.             }
135.     }
136.
137.     void Gauss_Seidel(int N, int M, double e, double fr,
   vector<vector<double> > &phi, vector<vector<double> > &phisup,
   vector<vector<double> > ap, vector<vector<double> > ae,
   vector<vector<double> > aw, vector<vector<double> > an,
   vector<vector<double> > as, vector<vector<double> > b)
138.     {
139.             //                      C�LCUL ITERATIU
140.
141.                     double dif=1; //valor inicialitzat major a la
   precisi� escollida 'e' perqu� entri al 'while'
142.                     int it=0;
143.
144.                     while(dif>e)
145.                     {
146.
147.
148.                             dif=0; //Per perdre el valor inicialitzat
149.
150.
151.                             //S'assignen els nous valors suposats amb
   factor de relaxaci�
152.                             //Nom�s t� sentit despr�s d'haver fet la 1a
   iteraci�
153.                             if(it>0)
154.                             {
155.                                     for(int j=0;j<=(M+1);j++)
156.                                     {
```

```
157.                                                    for(int i=0;i<=(N+1);i++)
158.                                                    {
159.                                                            //Nom?s es suposa un
   nou valor si la difer?ncia ?s m?s gran a l'error adm?s
160.                                                            if((fabs(phi[i][j]-
   phisup[i][j]))>e)
161.                                                            {
162.                                                                    phi[i][j]=phi
   sup[i][j]+fr*(phi[i][j]-phisup[i][j]);
163.                                                            }
164.                                                    }
165.                                            }
166.                                    }
167.
168.                                    //Es guarda el nou mapa de la variable phi
   suposat
169.
170.                                    for(int j=0;j<=(M+1);j++)
171.                                    {
172.                                            for(int i=0;i<=(N+1);i++)
173.                                            {
174.                                                    phisup[i][j]=phi[i][j];
175.
176.                                            }
177.
178.                                    }
179.
180.
181.                                    //              ESTRUCTURA DE C?LCUL
   PRINCIPAL: M?TODE GAUSS-SEIDEL
182.
183.                                    //C?lcul de tots els valors de la variable
   phi
184.
185.                                    //Nodes interns
186.                                    for(int j=1;j<=M;j++)
187.                                    {
188.                                            for(int i=1;i<=N;i++)
189.                                            {
190.                                                     phi[i][j]=(ae[i][j]*phi[i+1]
   [j]+aw[i][j]*phi[i-1][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-
   1]+b[i][j])/ap[i][j]; //Condici? general
191.                                            }
192.
193.                                    }
194.
195.
196.                                    //S'assigna el valor dels nodes dels v?rtexs
   com a promig dels del voltant
197.                                    phi[0][M+1]=(phi[1][M+1]+phi[0][M])/2; //Supe
   rior esquerre
198.                                    phi[0][0]=(phi[1][0]+phi[0][1])/2; //Inferior
   esquerre
199.                                    phi[N+1][M+1]=(phi[N][M+1]+phi[N+1][M])/2; //
   Superior dret
200.                                    phi[N+1][0]=(phi[N+1][1]+phi[N][0])/2; //Supe
   rior dret
201.
202.
203.
```

```
204.                                //Es calculen les difer?ncies i s'assigna la
   m?xima a la variable 'dif'
205.                                for(int j=0;j<=(M+1);j++)
206.                                {
207.                                        for(int i=0;i<=(N+1);i++)
208.                                        {
209.                                                maxdif(phi[i][j],
   phisup[i][j], dif);
210.                                        }
211.                                }
212.
213.                                it++;
214.                        } //fi 'while'
215.
216.                        cout<<it<<endl;
217.
218.        }
219.
220.
221.        int main()
222.        {
223.
224.                //                              2. C?LCULS PREVIS
225.
226.                const double deltax=Lx/N, deltay=Ly/M; //dist?ncia 'x' i 'y'
   dels VC
227.                vector <vector<double> > X(N+2,vector<double> (M+2)); //vecto
   rs de localitzaci? dels nodes
228.                vector <vector<double> > Y(X);
229.                vector<double>  Xc(N+1), Yc(M+1);
230.
231.
232.                //Creaci? de la malla
233.                malla(Lx, Ly, N, M, X, deltax, Y, deltay, Xc, Yc);
234.
235.
236.                vector <vector<double> > u(X), v(X); //vectors de les
   components horitz. i vert. de la velocitat
237.
238.                //Introducci? del camp de velocitats
239.                v_horitzontal(u, vel, angle);
240.                v_vertical(v, vel, angle);
241.
242.
243.                //                              3. COEFICIENTS DE
   DISCRETITZACI?
244.
245.                //Es calculen tots els coefs. de discretitzaci?
246.                vector <vector<double> > ae(X),aw(X),an(X),as(X),ap(X),b(X);
247.                double De,Dw,Dn,Ds,Fe,Fw,Fn,Fs,Pe,Pw,Pn,Ps;
248.
249.                for(int j=0;j<=(M+1);j++)
250.                {
251.                        for(int i=0;i<=(N+1);i++)
252.                        {
253.                                b[i][j]=0;
254.                        }
255.                }
256.
```

```
257.            //Els nodes dels v?rtexs no tenen coeficients de discr.
   (tindran un valor phi promig dels del voltant)
258.
259.            for(int j=1;j<=M;j++)
260.            {
261.                    for(int i=1;i<=N;i++)
262.                    {
263.                            //Cara est
264.                            De=gamma*deltay/fabs(X[i+1][j]-X[i][j]);
265.                            Fe=rho*u[i][j]*deltay;
266.                            Pe=Fe/De;
267.
268.                            ae[i][j]=De*A_P(Pe)+max(-Fe,0);
269.
270.
271.                            //Cara oest
272.                            Dw=gamma*deltay/fabs(X[i][j]-X[i-1][j]);
273.                            Fw=rho*u[i][j]*deltay;
274.                            Pw=Fw/Dw;
275.
276.                            aw[i][j]=Dw*A_P(Pw)+max(Fw,0);
277.
278.                            //Cara nord
279.                            Dn=gamma*deltax/fabs(Y[i][j+1]-Y[i][j]);
280.                            Fn=rho*v[i][j]*deltax;
281.                            Pn=Fn/Dn;
282.
283.                            an[i][j]=Dn*A_P(Pn)+max(-Fn,0);
284.
285.                            //Cara sud
286.                            Ds=gamma*deltax/fabs(Y[i][j]-Y[i][j-1]);
287.                            Fs=rho*v[i][j]*deltax;
288.                            Ps=Fs/Ds;
289.
290.                            as[i][j]=Ds*A_P(Ps)+max(Fs,0);
291.
292.                            //Node P
293.                            ap[i][j]=ae[i][j]+aw[i][j]+an[i][j]+as[i][j];
294.
295.                            //Terme independent
296.                            //?s nul si no hi ha terme font i no ?s
   transitori
297.
298.                    }
299.
300.            }
301.
302.
303.
304.
305.
306.            for(int i=1;i<=N;i++)
307.            {
308.                    ap[i][M+1]=1; ae[i][M+1]=0; aw[i][M+1]=0; an[i][M+1]=
   0; as[i][M+1]=1; b[i][M+1]=0; //Nodes superiors
309.                    ap[i][0]=1; ae[i][0]=0; aw[i][0]=0; an[i][0]=1; as[i]
   [0]=0; b[i][0]=0; //Nodes inferiors
310.            }
311.
312.
313.            //                              4. PROGRAMA PRINCIPAL
```

```
314.
315.            vector <vector<double> > phi(X),phisup(X);
316.
317.            //S'aplicaran les condicions de contorn de Dirichlet
318.
319.
320.            for(int j=1;j<=M;j++)
321.            {
322.                    phi[0][j]=phi1; //Condici? esquerra
323.                    phi[N+1][j]=phi2; //Condici? dreta
324.            }
325.
326.            for(int i=1;i<=N;i++)
327.            {
328.                    phi[i][M+1]=phi1; //Condici? superior
329.                    phi[i][0]=phi2; //Condici? inferior
330.            }
331.
332.
333.            //                      4.1 Valors suposats inicialment
334.
335.            //Se suposa la zona esquerra amb el valor de la CC esquerra
336.            for(int j=1;j<=M;j++)
337.            {
338.                    for(int i=1;i<=(N/2);i++)
339.                    {
340.                            phi[i][j]=phi1;
341.                    }
342.            }
343.
344.            //Se suposa la zona dreta amb el valor de la CC dreta
345.            for(int j=1;j<=M;j++)
346.            {
347.                    for(int i=(N/2+1);i<=N;i++)
348.                    {
349.                            phi[i][j]=phi2;
350.                    }
351.            }
352.
353.
354.            //S'assigna el valor dels nodes dels v?rtexs com a promig
   dels del voltant
355.            phi[0][M+1]=(phi[1][M+1]+phi[0][M])/2; //Superior esquerre
356.            phi[0][0]=(phi[1][0]+phi[0][1])/2; //Inferior esquerre
357.            phi[N+1][M+1]=(phi[N][M+1]+phi[N+1][M])/2; //Superior dret
358.            phi[N+1][0]=(phi[N][0]+phi[N+1][1])/2; //Inferior dret
359.
360.
361.
362.
363.
364.            //                      4.2 SOLVER: GAUSS-SEIDEL
365.            Gauss_Seidel(N, M, e, fr, phi, phisup, ap, ae, aw, an, as,
   b);
366.
367.
368.
369.            //              5. IMPRESSI? RESULTATS
370.
371.            ofstream myfile;
```

```
372.          myfile.open ("dades.txt");
373.
374.              for(int j=0;j<=(M+1);j++)
375.              {
376.                      for(int i=0;i<=(N+1);i++)
377.                      {
378.                              myfile<<phi[i][j]<<" ";
379.                      }
380.
381.                      myfile<<endl;
382.              }
383.
384.         myfile.close();
385.
386.      }
```

2.4 Cas 4

```
1. //      Convection-diffusion equation by CTTC - Case 4
2.
3. //      Code implemented by student Mario Luna L?pez
4.
5. #include<iostream>
6. #include<iomanip>
7. #include<math.h>
8. #include <vector>
9. #include <fstream>
10.      using namespace std;
11.
12.
13.      //                                1. ENTRADA DE DADES
14.
15.      //1.1 Dades F?siques
16.      const double Lx=1, Ly=1; //longituds del domini; Es un rectangle de
   2Lx?Ly
17.      const double rho=1000; //densitat del fluid
18.      const double gamma=1; //coeficient de difusi?
19.      const double alfa=10; //Constant del problema (sense sentit f?sic)
20.
21.      //1.2 Dades num?riques
22.      int dens_malla=200;  //Densitat de malla.
23.      const int N=2*dens_malla+1, M=dens_malla; //n?mero de nodes
   interiors en direcci? 'x' i 'y'
24.      //Nota: els nodes en 'x' han de ser imparells perqu? es col?loqui un
   node en x=0
25.      const double e=1e-10;//precisi? escollida per la resoluci?
26.      const double fr=1.1; //factor de relaxaci?
27.
28.
29.
30.
31.      void maxdif(double phi, double phisup, double &dif)
32.      {
33.              //Assigna el valor de la difer?ncia entre valor calculat i
   valor suposat a la variable 'dif' (si ?s major que aquesta)
34.              if(fabs(phi-phisup)>dif)    dif=fabs(phi-phisup);
35.      }
36.
37.      double max(double a, double b)
38.      {
39.              //Retorna el valor m?xim entre 2 valors (comptant el signe)
40.              if(a>b) return a;
41.              else return b;
42.      }
43.
44.
45.      void malla(double Lx, double Ly, double N, double M,
   vector<vector<double> > &X, double deltax,
   vector<vector<double> > &Y, double deltay)
46.      {
47.              //Construeix la malla 2D rectangular del problema
48.
49.
```

```
50.             //El seg?ent 'for' s'encarreguen d'emmagatzemar les
   coordenades dels nodes
51.             //Cal tenir en compte les coordenades especials com les dels
   nodes del contorn
52.             //Tamb? s?n especials els nodes separats delta/2 del contorn
53.
54.             //Coordenades x dels nodes
55.
56.             for(int j=0;j<=(M+1);j++)
57.             {
58.                     for(int i=0;i<=(N+1);i++)
59.                     {
60.                             if(i==0) X[i][j]=-Lx;
61.
62.                             else if(i==1) X[i][j]= deltax/2 + X[i-1][j];
63.
64.                             else if(i<=N) X[i][j]= deltax + X[i-1][j];
65.
66.                             else X[i][j]=Lx;
67.
68.                     }
69.             }
70.
71.             //Coordenades y dels nodes
72.             for(int j=0;j<=(M+1);j++)
73.             {
74.                     for(int i=0;i<=(N+1);i++)
75.                     {
76.                             if(j==0) Y[i][j]=0;
77.
78.                             else if(j==1) Y[i][j]= deltay/2;
79.
80.                             else if(j<=M) Y[i][j]= deltay + Y[i][j-1];
81.
82.                             else Y[i][j]=Ly;
83.
84.                     }
85.             }
86.
87.     }
88.
89.     double A_P(double P)
90.     {
91.             //Calcula el coeficient "Ahead" segons l'esquema introdu?t
92.
93.             //Exponential
94.             if(fabs(P)<=1e-10) return 1;
95.             else return fabs(P)/(exp(fabs(P))-1);
96.     }
97.
98.
99.     double v_horitzontal(double x, double y)
100.    {
101.            //VELOCITATS CORRESPONENTS AL TRAVESSAR LA CARA DEL VC
102.            return 2*y*(1-pow(x,2));
103.    }
104.
105.    double v_vertical(double x, double y)
106.    {
107.            //VELOCITATS CORRESPONENTS AL TRAVESSAR LA CARA DEL VC
108.            return -2*x*(1-pow(y,2));
```

```
109.     }
110.
111.     void Gauss_Seidel(int N, int M, double e, double fr,
   vector<vector<double> > &phi, vector<vector<double> > &phisup,
   vector<vector<double> > ap, vector<vector<double> > ae,
   vector<vector<double> > aw, vector<vector<double> > an,
   vector<vector<double> > as, vector<vector<double> > b)
112.     {
113.             //                      C?LCUL ITERATIU
114.
115.                     double dif=1; //valor inicialitzat major a la
   precisi? escollida 'e' perqu? entri al 'while'
116.                     int it=0;
117.
118.                     while(dif>e)
119.                     {
120.
121.
122.                             dif=0; //Per perdre el valor inicialitzat
123.
124.
125.                             //S'assignen els nous valors suposats amb
   factor de relaxaci?
126.                             //Nom?s t? sentit despr?s d'haver fet la 1a
   iteraci?
127.                             if(it>0)
128.                             {
129.                                     for(int j=0;j<=(M+1);j++)
130.                                     {
131.                                             for(int i=0;i<=(N+1);i++)
132.                                             {
133.                                                     //Nom?s es suposa un
   nou valor si la difer?ncia ?s m?s gran a l'error adm?s
134.                                                     if((fabs(phi[i][j]-
   phisup[i][j]))>e)
135.                                                     {
136.                                                             phi[i][j]=phi
   sup[i][j]+fr*(phi[i][j]-phisup[i][j]);
137.                                                     }
138.                                             }
139.                                     }
140.                             }
141.
142.                             //Es guarda el nou mapa de la variable phi
   suposat
143.
144.                             for(int j=0;j<=(M+1);j++)
145.                             {
146.                                     for(int i=0;i<=(N+1);i++)
147.                                     {
148.                                             phisup[i][j]=phi[i][j];
149.
150.                                     }
151.
152.                             }
153.
154.
155.                             //              ESTRUCTURA DE C?LCUL
   PRINCIPAL: M?TODE GAUSS-SEIDEL
156.
```

```
157.                                //C?lcul de tots els valors de la variable
   phi
158.
159.                                //Nodes interns
160.                                for(int j=1;j<=M;j++)
161.                                {
162.                                        for(int i=1;i<=N;i++)
163.                                        {
164.                                                phi[i][j]=(ae[i][j]*phi[i+1]
   [j]+aw[i][j]*phi[i-1][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-
   1]+b[i][j])/ap[i][j]; //Condici? general
165.                                        }
166.
167.                                }
168.
169.
170.                                //Nodes sortida
171.                                for(int i=((N+1)/2+1);i<=N;i++)
172.                                {
173.                                        phi[i][0]=phi[i][1]; //Contorn
   inferior
174.                                }
175.
176.
177.
178.                                //Es calculen les difer?ncies i s'assigna la
   m?xima a la variable 'dif'
179.                                for(int j=0;j<=(M+1);j++)
180.                                {
181.                                        for(int i=0;i<=(N+1);i++)
182.                                        {
183.                                                maxdif(phi[i][j],
   phisup[i][j], dif);
184.                                        }
185.
186.                                }
187.
188.                                it++;
189.                        } //fi 'while'
190.
191.                        cout<<it<<endl;
192.
193.        }
194.
195.        //Interpola el valor d'una funci? en 1-D
196.        double interpolar(double x1, double phi1, double x2, double phi2, dou
   ble x)
197.        {
198.                return (phi2-phi1)*(x-x1)/(x2-x1)+phi1;
199.        }
200.
201.        int trobar_punts_contorn(vector <vector<double> > X, double pos, int
   N)
202.        {
203.                bool trobat=false;
204.                int i=0;
205.
206.                while(trobat!=true)
207.                {
208.                        if(X[i][0]<pos && X[i+1][0]>pos)
```

```
209.                    {
210.                            trobat=true; //Si la posici? es troba entre
   dos punts consecutius de la malla, es guarda la 'i' dels punts
211.                    }
212.
213.                    i++;
214.            }
215.
216.            i--;//Ja que si no donaria una posici? m?s pel 'i++'
217.            return i;
218.    }
219.
220.    int main()
221.    {
222.
223.            //                              2. C?LCULS PREVIS
224.
225.            const double deltax=2*Lx/N, deltay=Ly/M; //dist?ncia 'x' i
   'y' dels VC
226.            //Nota: es multiplica per 2 ja que la dist?ncia total ?s Lx
   a l'esquerra i Lx a la dreta (2Lx)
227.            vector <vector<double> > X(N+2,vector<double> (M+2)); //vecto
   rs de localitzaci? dels nodes
228.            vector <vector<double> > Y(X);
229.
230.
231.            //Creaci? de la malla
232.            malla(Lx, Ly, N, M, X, deltax, Y, deltay); //OK
233.
234.            ofstream myfile;
235.         myfile.open ("mallax10.txt");
236.
237.            for(int j=0;j<=(M+1);j++)
238.            {
239.                    for(int i=0;i<=(N+1);i++)
240.                    {
241.                            myfile<<X[i][j]<<" ";
242.                    }
243.
244.                    myfile<<endl;
245.            }
246.
247.        myfile.close();
248.
249.        ofstream myfile2;
250.         myfile2.open ("mallay10.txt");
251.
252.            for(int j=0;j<=(M+1);j++)
253.            {
254.                    for(int i=0;i<=(N+1);i++)
255.                    {
256.                            myfile2<<Y[i][j]<<" ";
257.                    }
258.
259.                    myfile2<<endl;
260.            }
261.
262.        myfile2.close();
263.
264.
```

```
265.
266.            //                              3. COEFICIENTS DE
   DISCRETITZACI?
267.
268.            //Es calculen tots els coefs. de discretitzaci?
269.            vector <vector<double> > ae(X),aw(X),an(X),as(X),ap(X),b(X);
   //coeficients de discretitzaci?
270.            double De,Dw,Dn,Ds,Fe,Fw,Fn,Fs,Pe,Pw,Pn,Ps;
271.            double xw,yw,xe,ye,xn,yn,xs,ys; //posici? de les cares del
   VC del node i,j
272.            double velw,vele,veln,vels; //variable auxiliar on s'assigna
   la velocitat corresponent a la cara tractada
273.
274.            for(int j=0;j<=(M+1);j++)
275.            {
276.                    for(int i=0;i<=(N+1);i++)
277.                    {
278.                            b[i][j]=0;
279.                    }
280.            }
281.
282.            //Els nodes dels v?rtexs no tenen coeficients de discr.
   (tindran un valor phi promig dels del voltant)
283.
284.            for(int j=1;j<=M;j++)
285.            {
286.                    for(int i=1;i<=N;i++)
287.                    {
288.                            //              CARA EST
289.                            xe=X[i][j]+deltax/2;
290.                            ye=Y[i][j];
291.                            vele=v_horitzontal(xe,ye);
292.
293.                            De=gamma*deltay/fabs(X[i+1][j]-X[i][j]);
294.                            Fe=rho*vele*deltay;
295.                            Pe=Fe/De;
296.
297.                            ae[i][j]=De*A_P(Pe)+max(-Fe,0);
298.
299.
300.
301.                            //              CARA OEST
302.                            xw=X[i][j]-deltax/2;
303.                            yw=Y[i][j];
304.                            velw=v_horitzontal(xw,yw);
305.
306.                            Dw=gamma*deltay/fabs(X[i][j]-X[i-1][j]);
307.                            Fw=rho*velw*deltay;
308.                            Pw=Fw/Dw;
309.
310.                            aw[i][j]=Dw*A_P(Pw)+max(Fw,0);
311.
312.
313.
314.                            //              CARA NORD
315.                            xn=X[i][j];
316.                            yn=Y[i][j]+deltay/2;
317.                            veln=v_vertical(xn,yn);
318.
319.                            Dn=gamma*deltax/fabs(Y[i][j+1]-Y[i][j]);
```

```
320.                                Fn=rho*veln*deltax;
321.                                Pn=Fn/Dn;
322.
323.                                an[i][j]=Dn*A_P(Pn)+max(-Fn,0);
324.
325.
326.                                //              CARA SUD
327.                                xs=X[i][j];
328.                                ys=Y[i][j]-deltay/2;
329.                                vels=v_vertical(xs,ys);
330.
331.                                Ds=gamma*deltax/fabs(Y[i][j]-Y[i][j-1]);
332.                                Fs=rho*vels*deltax;
333.                                Ps=Fs/Ds;
334.
335.                                as[i][j]=Ds*A_P(Ps)+max(Fs,0);
336.
337.
338.                                //              NODE P
339.                                ap[i][j]=ae[i][j]+aw[i][j]+an[i][j]+as[i][j];
340.
341.                                //Terme independent
342.                                //s nul si no hi ha terme font i no s
   transitori
343.
344.                        }
345.
346.                }
347.
348.
349.
350.
351.
352.                //                              4. PROGRAMA PRINCIPAL
353.
354.                vector <vector<double> > phi(X),phisup(X);
355.
356.                //S'aplicaran les condicions de contorn de Dirichlet
357.
358.
359.                for(int i=1;i<=((N+1)/2);i++)
360.                {
361.                        phi[i][0]=1+tanh((2*X[i][0]+1)*alfa); //Contorn
   inferior (entrada)
362.                }
363.
364.                for(int i=0;i<=(N+1);i++)
365.                {
366.                        phi[i][M+1]=1-tanh(alfa); //Contorn superior
367.                }
368.
369.                for(int j=0;j<=M;j++)
370.                {
371.                        phi[0][j]=1-tanh(alfa); //Contorn esquerre
372.                        phi[N+1][j]=1-tanh(alfa); //Contorn dret
373.                }
374.
375.
376.
377.                //                      4.1 Valors suposats inicialment
378.
```

```
379.             //Se suposa els nodes interns amb valor 0
380.             for(int j=1;j<=M;j++)
381.             {
382.                     for(int i=1;i<=N;i++)
383.                     {
384.                             phi[i][j]=0;
385.                     }
386.             }
387.
388.             //Se suposa el contorn de sortida amb valor 0
389.             for(int i=((N+1)/2+1);i<=N;i++)
390.             {
391.                     phi[i][0]=0;
392.             }
393.
394.
395.
396.             //                       4.2 SOLVER: GAUSS-SEIDEL
397.             Gauss_Seidel(N, M, e, fr, phi, phisup, ap, ae, aw, an, as,
    b);
398.
399.
400.
401.             //              5. IMPRESSI RESULTATS
402.
403.             ofstream myfile3;
404.          myfile3.open ("dades.txt");
405.
406.             for(int j=0;j<=(M+1);j++)
407.             {
408.                     for(int i=0;i<=(N+1);i++)
409.                     {
410.                             myfile3<<phi[i][j]<<" ";
411.                     }
412.
413.                     myfile3<<endl;
414.             }
415.
416.         myfile3.close();
417.
418.
419.             //Impressi dels valors als punts demanats
420.
421.             int ix;
422.             double phix;
423.
424.             cout<<setprecision(3)<<fixed;
425.             for(double pos=0;pos<=1;pos=pos+0.1)
426.             {
427.                     ix=trobar_punts_contorn(X, pos, N);
428.                     phix=interpolar(X[ix][0], phi[ix][0], X[ix+1][0],
    phi[ix+1][0], pos);
429.                     cout<<"x="<<pos<<"     phi="<<phix<<"     es troba
    entre x="<<X[ix][0]<<" i  x="<<X[ix+1][0]<<endl;
430.             }
431.
432.     }
```

3 Equacions *Navier-Stokes*

```
1. //      Driven Cavity problem by CTTC
2. //      Fractional Step Method applied to solve the Navier-Stokes equations
3. //      Incompressible flow
4.
5. //      Code implemented by student Mario Luna L?pez
6.
7. #include<iostream>
8. #include<iomanip>
9. #include<math.h>
10.      #include <vector>
11.      #include <fstream>
12.      using namespace std;
13.
14.
15.      //                                1. ENTRADA DE DADES
16.
17.      //1.1 Dades F?siques
18.      const double Lx=1, Ly=1; //longituds del domini
19.      const double rho=3200; //densitat del fluid
20.      const double u_ref=1; // velocitat de refer?ncia de la paret
   superior
21.      const double mu_ref=1; // viscositat din?mica de refer?ncia
22.
23.      //1.2 Dades num?riques
24.      const int densitat_malla=20;
25.      const int N=2*densitat_malla, M=N; //n?mero de nodes interiors en
   direcci? 'x' i 'y'
26.      //Nota: N i M cal que siguin PARELLS!
27.      const double k=0.001; //factor d'estretament
28.      const double e=1e-8;//precisi? escollida per la resoluci?
29.      const double fr=1.1; //factor de relaxaci?
30.      const double e_temp=1e-3;//difer?ncia m?xima de velocitats entre
   time-step admesa per considerar estat estacionari (respecte el temps)
31.
32.
33.      //Assigna el valor de la difer?ncia entre valor calculat i valor
   suposat a la variable 'dif' (si ?s major que aquesta)
34.      void maxdif(double phi, double phisup, double &dif) //Validada
35.      {
36.
37.              if(fabs(phi-phisup)>dif)     dif=fabs(phi-phisup);
38.      }
39.
40.      //      Retorna el valor m?xim entre 2 valors (comptant el signe)
41.      double max(double a, double b) //Validada
42.      {
43.
44.              if(a>b) return a;
45.              else return b;
46.      }
47.
48.      //      Retorna el valor m?nim entre 2 valors (comptant el signe)
49.      double min(double a, double b) //Validada
50.      {
51.
```

```
52.             if(a<b) return a;
53.             else return b;
54.     }
55.
56.     //      Generaci□ de la malla (malla 2D rectangular)
57.     void malla(double Lx, double Ly, double N, double M, double k,
   vector<vector<double> > &Xc, vector<vector<double> > &Yc,
   vector<vector<double> > &X, vector<vector<double> > &Y,
   vector<vector<double> > &Xu, vector<vector<double> > &Yu,
   vector<vector<double> > &Xv, vector<vector<double> > &Yv) //Validada
58.     {
59.
60.
61.
62.             //El seg□ent 'for' s'encarreguen d'emmagatzemar les cares
   dels VC
63.             //La malla □s no uniforme, de distribuci□ hiperb□lica
64.
65.             //              1.                      COORDENADES X VC
66.
67.             double x1=0;
68.             double x2=Lx/2;
69.             double s;
70.
71.             // 1a meitat
72.             for(int j=0;j<=(M);j++)
73.             {
74.                     for(int i=0;i<=(N)/2;i++)
75.                     {
76.                             s=1+tanh(k*(i/((N)/2)-1))/tanh(k);
77.                             Xc[i][j]=x1+s*(x2-x1);
78.                     }
79.             }
80.
81.
82.             int i_aux;
83.             for(int j=0;j<=(M);j++)
84.             {
85.                     for(int i=((N)/2+1);i<=(N);i++)
86.                     {
87.                             if(i==((N)/2+1)) i_aux=i-1;
88.                             else i_aux--;
89.                             Xc[i][j]=Xc[i-1][j]+Xc[i_aux][j]-Xc[i_aux-
   1][j];
90.                     }
91.             }
92.
93.
94.             //              2.                      COORDENADES Y VC
95.
96.             double y1=0;
97.             double y2=Ly/2;
98.
99.             for(int j=0;j<=(M)/2;j++)
100.            {
101.                    for(int i=0;i<=(N);i++)
102.                    {
103.                            s=1+tanh(k*(j/((M)/2)-1))/tanh(k);
104.                            Yc[i][j]=y1+s*(y2-y1);
105.                    }
106.            }
```

```
107.
108.            int j_aux;
109.
110.            for(int i=0;i<=(N);i++)
111.            {
112.                    for(int j=((M)/2+1);j<=(M);j++)
113.                    {
114.                            if(j==((M)/2+1)) j_aux=j-1;
115.                            else j_aux--;
116.
117.                            Yc[i][j]=Yc[i][j-1]+Yc[i][j_aux]-Yc[i][j_aux-
    1];
118.                    }
119.            }
120.
121.
122.            //              3.                      COORDENADES NODES
123.
124.
125.            //Contorns esquerre i dret
126.            for(int j=0;j<=(M+1);j++)
127.            {
128.                    X[0][j]=0; //Contorn esquerre
129.                    X[N+1][j]=Lx; //Contorn dret
130.            }
131.
132.            //Contorns inferior i superior
133.            for(int i=0;i<=(N+1);i++)
134.            {
135.                    Y[i][0]=0; //Contorn inferior
136.                    Y[i][M+1]=Ly; //Contorn superior
137.            }
138.
139.            //              Coordenades X
140.            for(int j=0;j<=(M+1);j++)
141.            {
142.                    for(int i=1;i<=(N);i++) //Es fixa j=0 pels índexs de
    les cares dels VC ja que la distribució de les Xc és la mateixa per
    qualsevol j
143.                    { //Si no es fes aixó, per j=M+1 donaria problemes
    ja que les cares dels VC no tenen aquest índex (només fins j=M)
144.                            X[i][j]=Xc[i-1][0]+(Xc[i][0]-Xc[i-1][0])/2;
145.                    }
146.            }
147.
148.
149.            //              Coordenades Y
150.            for(int j=1;j<=(M);j++)
151.            {
152.                    for(int i=0;i<=(N+1);i++) //Es fixa i=0 pels índexs
    de les cares dels VC ja que la distribució de les Yc és la mateixa per
    qualsevol i
153.                    { //Si no es fes aixó, per i=N+1 donaria problemes
    ja que les cares dels VC no tenen aquest índex (només fins i=N)
154.                            Y[i][j]=Yc[0][j-1]+(Yc[0][j]-Yc[0][j-1])/2;
155.                    }
156.            }
157.
158.
159.
```

```
160.
161.
162.            // MALLA VELOCITAT u
163.            for(int j=0;j<=(M+1);j++)
164.            {
165.                    for(int i=0;i<=(N);i++)
166.                    {
167.                            Xu[i][j]=Xc[i][0];
168.                            Yu[i][j]=Y[0][j];
169.                    }
170.            }
171.
172.            // MALLA VELOCITAT v
173.            for(int j=0;j<=(M);j++)
174.            {
175.                    for(int i=0;i<=(N+1);i++)
176.                    {
177.                            Xv[i][j]=X[i][0];
178.                            Yv[i][j]=Yc[0][j];
179.                    }
180.            }
181.    }
182.
183.    //      Solver Gauss-Seidel
184.    void Gauss_Seidel(int N, int M, double e, double fr,
   vector<vector<double> > &phi, const vector<vector<double> > &ap, const vect
   or<vector<double> > &ae, const vector<vector<double> > &aw, constvector<vec
   tor<double> > &an, const vector<vector<double> > &as, const vector<vector<d
   ouble> > &b)
185.    {
186.            //                      C�LCUL ITERATIU
187.
188.                    vector<vector<double> > phisup(phi);
189.                    double dif=1; //valor inicialitzat major a la
   precisi� escollida 'e' perqu� entri al 'while'
190.                    int it=0;
191.
192.                    while(dif>e)
193.                    {
194.
195.
196.                            dif=0; //Per perdre el valor inicialitzat
197.
198.
199.                            //S'assignen els nous valors suposats amb
   factor de relaxaci�
200.                            //Nom�s t� sentit despr�s d'haver fet la 1a
   iteraci�
201.                            if(it>0)
202.                            {
203.                                    for(int j=0;j<=(M+1);j++)
204.                                    {
205.                                            for(int i=0;i<=(N+1);i++)
206.                                            {
207.                                                    //Nom�s es suposa un
   nou valor si la difer�ncia �s m�s gran a l'error adm�s
208.                                                    if((fabs(phi[i][j]-
   phisup[i][j]))>e)
209.                                                    {
```

```
210.                                                                    phi[i][j]=phi
   sup[i][j]+fr*(phi[i][j]-phisup[i][j]);
211.                                                            }
212.                                                    }
213.                                            }
214.                                    }
215.
216.                                    //Es guarda el nou mapa de la variable phi
   suposat
217.
218.                                    for(int j=0;j<=(M+1);j++)
219.                                    {
220.                                            for(int i=0;i<=(N+1);i++)
221.                                            {
222.                                                    phisup[i][j]=phi[i][j];
223.
224.                                            }
225.
226.                                    }
227.
228.
229.                                    //              ESTRUCTURA DE C�LCUL
   PRINCIPAL: M�TODE GAUSS-SEIDEL
230.
231.                                    //C�lcul dels valors de la variable phi
232.
233.                                    //Nodes interns
234.                                    for(int j=1;j<=M;j++)
235.                                    {
236.                                            for(int i=1;i<=N;i++)
237.                                            {
238.                                                     phi[i][j]=(ae[i][j]*phi[i+1]
   [j]+aw[i][j]*phi[i-1][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-
   1]+b[i][j])/ap[i][j]; //Condici� general
239.                                            }
240.
241.                                    }
242.
243.
244.                                    //Nodes contorn inferior i superior
245.                                    for(int i=1;i<=N;i++)
246.                                    {
247.                                            phi[i][0]=(ae[i][0]*phi[i+1][0]+aw[i]
   [0]*phi[i-1][0]+an[i][0]*phi[i][1]+b[i][0])/ap[i][0]; //Contorn inferior
248.                                            phi[i][M+1]=(ae[i][M+1]*phi[i+1][M+1]
   +aw[i][M+1]*phi[i-
   1][M+1]+as[i][M+1]*phi[i][M]+b[i][M+1])/ap[i][M+1]; //Contorn superior
249.                                    }
250.
251.
252.                                    //Nodes contorn esquerre i dret
253.                                    for(int j=1;j<=M;j++)
254.                                    {
255.                                            phi[0][j]=(ae[0][j]*phi[1][j]+an[0][j
   ]*phi[0][j+1]+as[0][j]*phi[0][j-1]+b[0][j])/ap[0][j]; //Contorn esquerre
256.                                            phi[N+1][j]=(aw[N+1][j]*phi[N][j]+an[
   N+1][j]*phi[N+1][j+1]+as[N+1][j]*phi[N+1][j-
   1]+b[N+1][j])/ap[N+1][j]; //Contorn superior
257.                                    }
258.
259.
```

```
260.
261.
262.
263.
264.                            //Es calculen les difer?ncies i s'assigna la
   m?xima a la variable 'dif'
265.                            for(int j=0;j<=(M+1);j++)
266.                            {
267.                                    for(int i=0;i<=(N+1);i++)
268.                                    {
269.                                            maxdif(phi[i][j],
   phisup[i][j], dif);
270.                                    }
271.                            }
272.
273.                            it++;
274.
275.                    } //fi 'while'
276.
277.                    cout<<it<<endl;
278.
279.                    //S'assigna el valor dels nodes dels v?rtexs com a
   promig dels del voltant
280.                    phi[0][M+1]=(phi[1][M+1]+phi[0][M])/2; //Superior
   esquerre
281.                    phi[0][0]=(phi[1][0]+phi[0][1])/2; //Inferior
   esquerre
282.                    phi[N+1][M+1]=(phi[N][M+1]+phi[N+1][M])/2; //Superior
   dret
283.                    phi[N+1][0]=(phi[N+1][1]+phi[N][0])/2; //Superior
   dret
284.
285.     }
286.
287.     // Solver TDMA (1-D)
288.     void TDMA(int N,
   vector<double> &phi, const vector<double> &aw, const vector<double> &ae, co
   nst vector<double> &b, const vector<double> &ap)
289.     {
290.
291.             vector<double> P(N+2);
292.             vector<double> R(N+2);
293.
294.             //      C?LCUL COEFICIENTS P i R
295.             for(int i=0; i<=(N+1);i++)
296.             {
297.                     if(i==0){ P[i]=ae[i]/ap[i]; R[i]=b[i]/ap[i];}
298.                     else if(i==(N+1)){ P[i]=0; R[i]=(b[i]+aw[i]*R[i-
   1])/(ap[i]-aw[i]*P[i-1]);}
299.                     else{ P[i]=ae[i]/(ap[i]-aw[i]*P[i-
   1]); R[i]=(b[i]+aw[i]*R[i-1])/(ap[i]-aw[i]*P[i-1]);}
300.             }
301.
302.             //      C?LCUL VARIABLE PHI
303.             for(int i=(N+1); i>=0;i--)
304.             {
305.                     if(i==(N+1)) phi[i]=R[i];
306.                     else phi[i]=P[i]*phi[i+1]+R[i];
307.             }
308.     }
```

```
309.
310.     // Solver Line-by-line (Gauss-Seidel i TDMA)
311.     void Line_by_line_solver(int N, int M, double e,
   vector<vector<double> > &phi, const vector<vector<double> > &aw, const vect
   or<vector<double> > &ae, const vector<vector<double> > &as, constvector<vec
   tor<double> > &an, const vector<vector<double> > &b, const vector<vector<do
   uble> > &ap)
312.     {
313.
314.
315.             vector<double> b_line(N+2), phi_line(N+2), ae_line(N+2),
   aw_line(N+2), ap_line(N+2); //1r escombrat
316.             vector<double> b_line_(M+2), phi_line_(M+2), as_line(M+2),
   an_line(M+2), ap_line_(M+2); //2n escombrat
317.             vector<vector<double> > phisup(phi);
318.             double dif=1;//valor inicialitzat major a la precisi?
   escollida 'e' perqu? entri al 'while'
319.             int it=0;
320.
321.             while(dif>e)
322.             {
323.                     dif=0;//Per perdre el valor inicialitzat
324.
325.                     //Es guarda el nou mapa de la variable phi
   suposat
326.                     for(int j=0;j<=(M+1);j++)
327.                     {
328.                             for(int i=0;i<=(N+1);i++)
329.                             {
330.                                     phisup[i][j]=phi[i][j];
331.                             }
332.                     }
333.
334.
335.
336.                     //              1r ESCOMBRAT: Escombrat de baix a
   dalt
337.                     for(int j=0; j<=(M+1);j++)
338.                     {
339.                             // 1. Es guarden els valors de la l?nia
340.                             for(int i=0; i<=(N+1);i++)
341.                             {
342.                                     phi_line[i]=phi[i][j];  ae_line[i]=ae
   [i][j];    aw_line[i]=aw[i][j];    ap_line[i]=ap[i][j];
343.                             }
344.
345.                             //      2. Es recalcula la b_line
346.                             if(j==0) //No nodes sud
347.                             {
348.                                     for(int i=0; i<=(N+1);i++) {b_line[i]
   =b[i][j]+an[i][j]*phi[i][j+1];}
349.                             }
350.                             else if(j==(M+1)) //No nodes nord
351.                             {
352.                                     for(int i=0; i<=(N+1);i++){b_line[i]=
   b[i][j]+as[i][j]*phi[i][j-1];}
353.                             }
354.                             else
355.                             {
356.                                     for(int i=0; i<=(N+1);i++){b_line[i]=
   b[i][j]+an[i][j]*phi[i][j+1]+as[i][j]*phi[i][j-1];}
```

```
357.                                }
358.
359.                                //      3. TDMA SOLVER
360.                                TDMA(N,phi_line,aw_line,ae_line,b_line,ap_lin
   e);
361.
362.
363.                                // 4. Es guarden els valors calculats
364.                                for(int i=0; i<=(N+1);i++){phi[i][j]=phi_line
   [i];}
365.                        }
366.
367.
368.
369.
370.                        //              2n ESCOMBRAT:Escombrat d'esquerra a
   dreta
371.                        for(int i=0; i<=(N+1);i++)
372.                        {
373.                                // 1. Es guarden els valors de la l•nia
374.                                for(int j=0; j<=(M+1);j++)
375.                                {
376.                                        phi_line_[j]=phi[i][j]; as_line[j]=as
   [i][j];    an_line[j]=an[i][j];    ap_line_[j]=ap[i][j];
377.                                }
378.
379.                                //      2. Es recalcula la b_line
380.                                if(i==0) //No nodes oest
381.                                {
382.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+ae[i][j]*phi[i+1][j];}
383.                                }
384.                                else if(i==(N+1)) //No nodes est
385.                                {
386.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+aw[i][j]*phi[i-1][j];}
387.                                }
388.                                else
389.                                {
390.                                        for(int j=0; j<=(M+1);j++){b_line_[j]
   =b[i][j]+ae[i][j]*phi[i+1][j]+aw[i][j]*phi[i-1][j];}
391.                                }
392.
393.                                //      3. TDMA SOLVER
394.                                TDMA(M,phi_line_,as_line,an_line,b_line_,ap_l
   ine_);
395.
396.                                // 4. Es guarden els valors calculats
397.                                for(int j=0; j<=(M+1);j++){phi[i][j]=phi_line
   _[j];}
398.                        }
399.
400.
401.                        //Es calculen les difer•ncies i s'assigna la m•xima
   a la variable 'dif'
402.                                for(int j=0;j<=(M+1);j++)
403.                                {
404.                                        for(int i=0;i<=(N+1);i++)
405.                                        {
406.                                                maxdif(phi[i][j],
   phisup[i][j], dif);
```

```
407.                                    }
408.                                }
409.                        it++;
410.
411.                }//Fi while
412.
413.                cout<<"Nombre d'iteracions: "<<it<<endl<<endl;
414.
415.        }
416.
417.        //      Mostra per pantalla els coeficients introdu?ts
418.        void mostrar_coeficients(int N, int M, const vector<vector<double> >
    &aw, const vector<vector<double> > &ae, const vector<vector<double> > &as,
    const vector<vector<double> > &an, constvector<vector<double> > &b, const v
    ector<vector<double> > &ap)
419.        {
420.                        cout<<"Coeficients ap:"<<endl;
421.                        for(int j=0;j<=(M+1);j++)
422.                        {
423.                                for(int i=0;i<=(N+1);i++)
424.                                {
425.                                        cout<<ap[i][j]<<" ";
426.                                }
427.
428.                                cout<<endl;
429.                        }
430.                        cout<<endl<<endl;
431.
432.                        cout<<"Coeficients ae:"<<endl;
433.                        for(int j=0;j<=(M+1);j++)
434.                        {
435.                                for(int i=0;i<=(N+1);i++)
436.                                {
437.                                        cout<<ae[i][j]<<" ";
438.                                }
439.
440.                                cout<<endl;
441.                        }
442.                        cout<<endl<<endl;
443.
444.                        cout<<"Coeficients aw:"<<endl;
445.                        for(int j=0;j<=(M+1);j++)
446.                        {
447.                                for(int i=0;i<=(N+1);i++)
448.                                {
449.                                        cout<<aw[i][j]<<" ";
450.                                }
451.
452.                                cout<<endl;
453.                        }
454.                        cout<<endl<<endl;
455.
456.                        cout<<"Coeficients an:"<<endl;
457.                        for(int j=0;j<=(M+1);j++)
458.                        {
459.                                for(int i=0;i<=(N+1);i++)
460.                                {
461.                                        cout<<an[i][j]<<" ";
462.                                }
463.
464.                                cout<<endl;
```

```
465.                    }
466.                    cout<<endl<<endl;
467.
468.                    cout<<"Coeficients as:"<<endl;
469.                    for(int j=0;j<=(M+1);j++)
470.                    {
471.                            for(int i=0;i<=(N+1);i++)
472.                            {
473.                                    cout<<as[i][j]<<" ";
474.                            }
475.
476.                            cout<<endl;
477.                    }
478.                    cout<<endl<<endl;
479.
480.                    cout<<"Coeficients bp:"<<endl;
481.                    for(int j=0;j<=(M+1);j++)
482.                    {
483.                            for(int i=0;i<=(N+1);i++)
484.                            {
485.                                    cout<<b[i][j]<<" ";
486.                            }
487.
488.                            cout<<endl;
489.                    }
490.                    cout<<endl<<endl;
491.    }
492.
493.    //      Guarda la malla en fitxers .txt
494.    void guardar_malla(int N, int M, const vector<vector<double> > &X, co
   nst vector<vector<double> > &Y, const vector<vector<double> > &Xc, const ve
   ctor<vector<double> > &Yc, const vector<vector<double> > &Xu, const vector<
   vector<double> > &Yu, const vector<vector<double> > &Xv, const vector<vecto
   r<double> > &Yv)
495.    {
496.            ofstream myfile;
497.        myfile.open ("malla_nodes_x.txt");
498.        //myfile.open (s2.c_str());
499.
500.            for(int i=0;i<=(N+1);i++)
501.            {
502.                    for(int j=0;j<=(M+1);j++)
503.                    {
504.                            myfile<<X[i][j]<<" ";
505.                    }
506.
507.                    myfile<<endl;
508.            }
509.
510.       myfile.close();
511.
512.       ofstream myfile2;
513.        myfile2.open ("malla_nodes_y.txt");
514.
515.            for(int i=0;i<=(N+1);i++)
516.            {
517.                    for(int j=(M+1);j>=0;j--)
518.                    {
519.                            myfile2<<Y[i][j]<<" ";
520.                    }
521.
```

```
522.                     myfile2<<endl;
523.               }
524.
525.          myfile2.close();
526.
527.          ofstream myfile3;
528.               myfile3.open ("malla_cares_x.txt");
529.
530.               for(int i=0;i<=(N);i++)
531.               {
532.                        for(int j=0;j<=(M);j++)
533.                        {
534.                                 myfile3<<Xc[i][j]<<" ";
535.                        }
536.
537.                        myfile3<<endl;
538.               }
539.
540.          myfile3.close();
541.
542.
543.          ofstream myfile4;
544.               myfile4.open ("malla_cares_y.txt");
545.
546.               for(int i=0;i<=(N);i++)
547.               {
548.                        for(int j=(M);j>=0;j--)
549.                        {
550.                                 myfile4<<Yc[i][j]<<" ";
551.                        }
552.
553.                        myfile4<<endl;
554.               }
555.
556.          myfile4.close();
557.
558.
559.          ofstream myfile13;
560.               myfile13.open ("malla_velocitat_xu.txt");
561.
562.               for(int i=0;i<=(N);i++)
563.               {
564.                        for(int j=0;j<=(M+1);j++)
565.                        {
566.                                 myfile13<<Xu[i][j]<<" ";
567.                        }
568.
569.                        myfile13<<endl;
570.               }
571.
572.          myfile13.close();
573.
574.          ofstream myfile14;
575.               myfile14.open ("malla_velocitat_yu.txt");
576.
577.               for(int i=0;i<=(N);i++)
578.               {
579.                        for(int j=(M+1);j>=0;j--)
580.                        {
581.                                 myfile14<<Yu[i][j]<<" ";
582.                        }
```

```
583.
584.                     myfile14<<endl;
585.             }
586.
587.         myfile14.close();
588.
589.         ofstream myfile15;
590.             myfile15.open ("malla_velocitat_xv.txt");
591.
592.             for(int i=0;i<=(N+1);i++)
593.             {
594.                     for(int j=0;j<=(M);j++)
595.                     {
596.                             myfile15<<Xv[i][j]<<" ";
597.                     }
598.
599.                     myfile15<<endl;
600.             }
601.
602.         myfile15.close();
603.
604.         ofstream myfile16;
605.             myfile16.open ("malla_velocitat_yv.txt");
606.
607.             for(int i=0;i<=(N+1);i++)
608.             {
609.                     for(int j=(M);j>=0;j--)
610.                     {
611.                             myfile16<<Yv[i][j]<<" ";
612.                     }
613.
614.                     myfile16<<endl;
615.             }
616.
617.         myfile16.close();
618.     }
619.
620.     //      Interpolaci? lineal (1-D) entre dos punts
621.     double interpolar(double x1, double phi1, double x2, double phi2, dou
    ble x)
622.     {
623.             return (phi2-phi1)*(x-x1)/(x2-x1)+phi1;         //Interpola
    el valor d'una funci? en 1-D
624.     }
625.
626.     //      Troba els punts x del voltant donada una posici? x (pos_x)
627.     int trobar_punts_x_contorn(const vector<vector<double> > &X, double p
    os_x)
628.     {
629.             bool trobat=false;
630.             int i=0;
631.
632.             while(trobat!=true)
633.             {
634.                     if(X[i][0]<=pos_x && X[i+1][0]>=pos_x)
635.                     {
636.                             trobat=true; //Si la posici? es troba entre
    dos punts consecutius de la malla, es guarda la 'i' dels punts
637.                     }
638.
639.                     i++;
```

```
640.             }
641.
642.             i--;//Ja que si no donaria una posici? m?s pel 'i++'
643.             return i;
644.     }
645.
646.     //      Troba els punts y del voltant donada una posici? y (pos_y)
647.     int trobar_punts_y_contorn(const vector<vector<double> > &Y, double p
   os_y)
648.     {
649.             bool trobat=false;
650.             int j=0;
651.
652.             while(trobat!=true)
653.             {
654.                     if(Y[0][j]<=pos_y && Y[0][j+1]>=pos_y)
655.                     {
656.                             trobat=true; //Si la posici? es troba entre
   dos punts consecutius de la malla, es guarda la 'i' dels punts
657.                     }
658.
659.                     j++;
660.             }
661.
662.             j--;//Ja que si no donaria una posici? m?s pel 'j++'
663.             return j;
664.     }
665.
666.     //      Retorna el valor d'una variable en un punt espec?fic del
   domini (pos_x, pos_y)
667.     double valor_punt(double pos_x, double pos_y, const vector<vector<dou
   ble> > &phi, const vector<vector<double> > &X, const vector<vector<double>
   > &Y) //Validada
668.     {
669.             //              1. Trobar les posicions dels punts del
   voltant
670.             int i,j;
671.             i=trobar_punts_x_contorn(X, pos_x);
672.             j=trobar_punts_y_contorn(Y, pos_y);
673.
674.             //              2. Interpolar entre els punts del voltant
675.
676.             //El punt del valor buscat es troba entre els punts 1, 2, 3 i
   4 (vegeu representaci? a sota)
677.             //---3----4----
678.             //---1----2----
679.
680.             double phi_inf, phi_sup;
681.
682.             //1r s'interpola entre els punts 1 i 2, trobant phi_inf
683.             phi_inf=interpolar(X[i][j], phi[i][j], X[i+1][j],
   phi[i+1][j], pos_x);
684.             //2n s'interpola entre els punts 3 i 4, trobant phi_sup
685.             phi_sup=interpolar(X[i][j+1], phi[i][j+1], X[i+1][j+1],
   phi[i+1][j+1], pos_x);
686.
687.             //El valor buscat ?s la interpolaci? entre phi_inf i phi_sup
688.             return interpolar(Y[i][j], phi_inf, Y[i][j+1], phi_sup,
   pos_y);
689.
```

```
690.      }
691.
692.      // Introducci? de les condicions de contorn de velocitat horitzontal
    u
693.      void u_cc(vector<vector<double> > &u, double u_ref, int N, int M)
694.      {
695.              //Contorn inferior i superior
696.              for(int i=0;i<=(N);i++)
697.              {
698.                      u[i][0]=0; //Contorn inferior
699.                      u[i][M+1]=u_ref; //Contorn superior
700.              }
701.
702.              //Contorn esquerre i dret
703.              for(int j=1;j<=(M);j++)
704.              {
705.                      u[0][j]=0; //Contorn esquerre
706.                      u[N][j]=0; //Contorn dret
707.              }
708.
709.      }
710.
711.      // Introducci? de les condicions de contorn de velocitat vertical v
712.      void v_cc(vector<vector<double> > &v, int N, int M)
713.      {
714.              //Contorn inferior i superior
715.              for(int i=0;i<=(N+1);i++)
716.              {
717.                      v[i][0]=0; //Contorn inferior
718.                      v[i][M]=0; //Contorn superior
719.              }
720.
721.              //Contorn esquerre i dret
722.              for(int j=1;j<=(M-1);j++)
723.              {
724.                      v[0][j]=0; //Contorn esquerre
725.                      v[N+1][j]=0; //Contorn dret
726.              }
727.
728.      }
729.
730.      //      Introducci? del camp de viscositat din?mica mu
731.      void mu_camp(vector<vector<double> > &mu, double mu_ref, int N, int M
    )
732.      {
733.              for(int j=0;j<=(M+1);j++)
734.              {
735.                      for(int i=0;i<=(N+1);i++)
736.                      {
737.                              mu[i][j]=mu_ref;
738.                      }
739.              }
740.      }
741.
742.      //      Esquema per les variables convectives
743.      double esquema_upwind(double phi_ant, double phi_post, double m_punt)
744.      {
745.              //phi_ant (anterior) i phi_post (posterior) indiquen un
    sentit creixent de la coordenada de phi_ant a phi_post
746.              //Aquest sentit ?s important ja que m_punt est? definit
    positiu en el sentit positiu de la coordenada
```

```
747.             if(m_punt>0) return phi_ant;
748.             else return phi_post;
749.     }
750.
751.     //      Introducci de les condicions de contorn als coeficients de
   pressi
752.     void coeficients_pressio_cc(int N, int M,
   vector<vector<double> > &ap, vector<vector<double> > &ae,
   vector<vector<double> > &aw, vector<vector<double> > &an,
   vector<vector<double> > &as, vector<vector<double> > &bp)
753.     {
754.             for(int j=1;j<=(M);j++)
755.             {
756.                     ap[0][j]=1; ae[0][j]=1; aw[0][j]=0; an[0][j]=0; as[0]
   [j]=0; bp[0][j]=0; //Contorn esquerre
757.                     ap[N+1][j]=1; ae[N+1][j]=0; aw[N+1][j]=1; an[N+1][j]=
   0; as[N+1][j]=0; bp[N+1][j]=0; //Contorn dret
758.             }
759.
760.             for(int i=1;i<=(N);i++)
761.             {
762.                     ap[i][0]=1; ae[i][0]=0; aw[i][0]=0; an[i][0]=1; as[i]
   [0]=0; bp[i][0]=0; //Contorn inferior
763.                     ap[i][M+1]=1; ae[i][M+1]=0; aw[i][M+1]=0; an[i][M+1]=
   0; as[i][M+1]=1; bp[i][M+1]=0; //Contorn superior
764.             }
765.
766.             //V rtexs
767.             ap[0][0]=1;             ap[0][M+1]=1;           ap[N+1][0]=1;
            ap[N+1][M+1]=1;
768.
769.     }
770.
771.     //      Introducci d'una pressi de refer ncia en un node
772.     void pressio_ref(int N, int M, vector<vector<double> > &ap,
   vector<vector<double> > &ae, vector<vector<double> > &aw,
   vector<vector<double> > &an, vector<vector<double> > &as,
   vector<vector<double> > &bp)
773.     {
774.             double p_ref=100000;
775.             //Es tria indicar una pressi de refer ncia en el node
   interior de l'extrem superior dret
776.             ae[N][M]=0;     aw[N][M]=0;     an[N][M]=0;     as[N][M]=0;
777.             ap[N][M]=1;
778.             bp[N][M]=p_ref;
779.     }
780.
781.     //      Calcula el deltat adient per a la integraci temporal segons
   la condici de Courant-Friedrichs-Levy
782.     double delta_temps(int N, int M, const vector<vector<double> > &X, co
   nst vector<vector<double> > &Y, const vector<vector<double> > &Xc, const ve
   ctor<vector<double> > &Yc, const vector<vector<double> > &u, const vector<v
   ector<double> > &v, double rho, const vector<vector<double> > &mu)
783.     {
784.             //Inicialitzaci a un valor alt per perdre'l despr s
785.             double deltatc=100, deltatd=100, deltat=100, deltat_aux=100;
786.
787.             //Deltatc: u
788.             for(int j=1;j<=(M);j++)
789.             {
```

```
790.                    for(int i=1;i<=(N-1);i++)
791.                    {
792.                            deltat_aux=fabs((X[i+1][j]-X[i][j])/u[i][j]);
793.                            deltatc=min(deltatc, deltat_aux);
794.                    }
795.            }
796.
797.            //Deltatc: v
798.            for(int j=1;j<=(M-1);j++)
799.            {
800.                    for(int i=1;i<=(N);i++)
801.                    {
802.                            deltat_aux=fabs((Y[i][j+1]-Y[i][j])/v[i][j]);
803.                            deltatc=min(deltatc, deltat_aux);
804.                    }
805.            }
806.
807.            //Deltatd
808.            for(int j=1;j<=M;j++)
809.            {
810.                    for (int i=1;i<=N;i++)
811.                    {
812.                            deltat_aux=fabs(rho*(Yc[i][j]-Yc[i][j-
    1])*(Xc[i][j]-Xc[i-1][j])/mu[i][j]);
813.                            deltatd=min(deltatd, deltat_aux);
814.                    }
815.            }
816.
817.
818.
819.            deltatc=0.35*deltatc;
820.            deltatd=0.20*deltatd;
821.            cout<<endl<<"deltatc="<<deltatc<<"
    deltatd="<<deltatd<<endl<<endl<<endl;
822.            deltat=min(deltatc, deltatd);
823.            return deltat;
824.    }
825.
826.    void velocitat_ppal(int N, int M, double u_ref,
    vector<vector<double> > &vel_u,
    vector<vector<double> > &vel_v, const vector<vector<double> > &u, const vec
    tor<vector<double> > &v)
827.    {
828.            //Nodes interns
829.            for(int j=1;j<=(M);j++)
830.            {
831.                    for(int i=1;i<=(N);i++)
832.                    {
833.                            vel_u[i][j]=(u[i][j]+u[i-1][j])/2;
834.                            vel_v[i][j]=(v[i][j]+v[i][j-1])/2;
835.                    }
836.            }
837.
838.            //Nodes contorn esquerre i dret
839.            for(int j=0;j<=(M);j++)
840.            {
841.                    vel_u[0][j]=0; vel_u[N+1][j]=0;
842.                    vel_v[0][j]=0; vel_v[N+1][j]=0;
843.            }
844.
845.            //Nodes contorn inferior i superior
```

```
846.             for(int i=0;i<=(N+1);i++)
847.             {
848.                     vel_u[i][0]=0; vel_u[i][M+1]=u_ref;
849.                     vel_v[i][0]=0; vel_v[i][M+1]=0;
850.             }
851.     }
852.
853.
854.     //      Retorna una variable (o posici�) normalitzada pels esquemes
   num�rics de convecci�-difusi�
855.     double var_normalitzada(double phi, double phiu, double phid)
856.     {
857.             return ((phi-phiu)/(phid-phiu));
858.     }
859.
860.     //      Esquema convectiu-difusiu SMART que retorna el valor
   normalitzat d'una variable a la cara
861.     double SMART(double phic, double xc, double xf)
862.     {
863.             if(phic>0 && phic<(xc/3)) return (-xf*(1-
   3*xc+2*xf)*phic/(xc*(xc-1)));
864.             else if(phic>(xc/3) && phic<(xc*(1+xf-
   xc)/xf)) return (xf*(xf-xc)/(1-xc)+xf*(xf-1)*phic/(xc*(xc-1)));
865.             else if(phic>(xc*(1+xf-xc)/xf) && phic<1) return 1;
866.             else return phic;
867.     }
868.
869.     //      Avaluaci� d'una variable a la cara d'un VC amb esquemes
   convectius-difusius
870.     double esquema_conveccio_difusio(double x, double phill, double xll,
   double phil, double xl, double phir, double xr, double phirr, double xrr, d
   ouble m_punt)
871.     {
872.             // Nomenclatura: 'll' left left, 'l' left, 'r' right, 'rr'
   right right; s�n els nodes que queden a esquerra i dreta del valor a la
   cara desitjat
873.
874.             double phic, xc, phif, xf; //variables normalitzades
875.
876.             if(m_punt>0)
877.             {
878.
879.                     if(phill==phir) return phil;
880.
881.                     // C�lcul valors adimensionals
882.                     phic=var_normalitzada(phil,phill,phir);         xc=va
   r_normalitzada(xl,xll,xr);
883.                                                             xf=var_normalitzada(x
   ,xll,xr);
884.
885.                     //      �s esquema
886.                     phif=SMART(phic,xc,xf);
887.
888.                     return (phif*(phir-phill)+phill); //Es retorna el
   valor dimensional
889.             }
890.
891.             else //m_punt<0
892.             {
893.                     if(phirr==phil) return phir;
```

```
894.                    //      C?lcul valors adimensionals
895.                    phic=var_normalitzada(phir,phirr,phil);         xc=va
   r_normalitzada(xr,xrr,xl);
896.                                                            xf=var_normalitzada(x
   ,xrr,xl);
897.
898.                    //      ?s esquema
899.                    phif=SMART(phic,xc,xf);
900.
901.                    return (phif*(phil-phirr)+phirr); //Es retorna el
   valor dimensional
902.            }
903.
904.    }
905.
906.    void funcions_corrent_u(vector<vector<double> > &f_corrent, int N, in
   t M, const vector<vector<double> > &Yc, const vector<vector<double> > &u)
907.    {
908.            for(int i=0;i<=(N);i++)
909.            {
910.                    for(int j=0;j<=(M);j++)
911.                    {
912.                            if(j==0) f_corrent[i][j]=0;
913.                            else f_corrent[i][j]=f_corrent[i][j-
   1]+u[i][j]*fabs(Yc[i][j]-Yc[i][j-1]);
914.                    }
915.            }
916.    }
917.
918.    int main()
919.    {
920.
921.            vector <vector<double> > X(N+2,vector<double> (M+2)),
   Y(X); //vectors de localitzaci? dels nodes
922.            vector <vector<double> > Xc(N+1,vector<double> (M+1)),
   Yc(Xc); //vectors dels VC (v?rtexs)
923.            vector <vector<double> > Xu(N+1,vector<double> (M+2)),
   Yu(Xu);
924.            vector <vector<double> > Xv(N+2,vector<double> (M+1)),
   Yv(Xv);
925.
926.            //Creaci? de la malla
927.            malla(Lx, Ly, N, M, k, Xc, Yc, X, Y, Xu, Yu, Xv, Yv);
928.
929.            //Guardar malla en fitxers .txt
930.            guardar_malla(N, M, X, Y, Xc, Yc, Xu, Yu, Xv, Yv);
931.
932.
933.
934.            vector <vector<double> > u(N+1,vector<double> (M+2)), u0(u);
935.            vector <vector<double> > v(N+2,vector<double> (M+1)), v0(v);
936.
937.            //Velocitats conegudes als contorns (condicions de contorn)
   per u i v (components x i y)
938.            u_cc(u,u_ref,N,M);
939.            v_cc(v,N,M);
940.
941.
942.            vector<vector<double> > mu(N+2,vector<double> (M+2));
```

```
943.            mu_camp(mu, mu_ref, N, M); //Les viscositats es defineixen a
   cada punt de la malla principal
944.
945.
946.
947.            //Camp de velocitats inicialment suposat (nodes interns)
948.            for(int j=1;j<=(M);j++)
949.            {
950.                    for(int i=1;i<=(N-1);i++)
951.                    {
952.                            u[i][j]=0;
953.                    }
954.            }
955.
956.
957.
958.            for(int j=1;j<=(M-1);j++)
959.            {
960.                    for(int i=1;i<=(N);i++)
961.                    {
962.                            v[i][j]=0;
963.                    }
964.            }
965.
966.
967.            //C?lcul volums control malles velocitat
968.            vector <vector<double> > omegax(N+1,vector<double> (M+2));
969.            vector <vector<double> > omegay(N+2,vector<double> (M+1));
970.
971.            for(int j=1;j<=(M);j++)
972.            {
973.                    for(int i=1;i<=(N-1);i++)
974.                    {
975.                            omegax[i][j]=(X[i+1][j]-X[i][j])*(Yc[i][j]-
   Yc[i][j-1]);
976.                    }
977.            }
978.
979.            for(int j=1;j<=(M-1);j++)
980.            {
981.                    for(int i=1;i<=(N);i++)
982.                    {
983.                            omegay[i][j]=(Y[i][j+1]-Y[i][j])*(Xc[i][j]-
   Xc[i-1][j]);
984.                    }
985.            }
986.
987.
988.            double deltat=0.0001, t=0;
989.
990.
991.            //                                                  FRACT
   IONAL STEP METHOD
992.
993.            double mu_e, mu_w, mu_n, mu_s, pos_x, pos_y; //variables
   viscositat
994.            double m_punt_e, m_punt_w, m_punt_n, m_punt_s; //variables
   flux m?ssic
995.            double ue, uw, un, us, ve, vw, vn, vs; //variables velocitats
   a les cares
```

```
996.            vector <vector<double> > Ru(N+1,vector<double> (M+2)),
   Ru0(Ru); //Ru instant n i Ru0 instant anterior n-1
997.            vector <vector<double> > Rv(N+2,vector<double> (M+1)),
   Rv0(Rv); //Rv instant n i Rv0 instant anterior n-1
998.            vector <vector<double> > uP(Ru), vP(Rv); //velocitats
   predictores
999.            double Ae, An; //rees de les cares dels VC
1000.           double dPE, dPW, dPN, dPS; //dist ncies entre nodes
1001.           vector <vector<double> > ap(N+2,vector<double> (M+2)),
   ae(ap), aw(ap), an(ap), as(ap), bp(ap); //Coeficients de discretitzaci
1002.           vector <vector<double> > p(N+2,vector<double> (M+2)); //Camp
   de pressions
1003.           double dif_u=0, dif_v=0, dif=1; //variables de les
   difer ncies de velocitat entre pas temporal
1004.
1005.
1006.    while(dif>e_temp)
1007.    {
1008.
1009.
1010.
1011.
1012.           if(t>0) //Despr s de l'instant inicial
1013.           {
1014.                   //C lcul deltat escaient
1015.                   deltat=delta_temps(N, M, X, Y, Xc, Yc, u, v, rho,
   mu);
1016.           }
1017.
1018.
1019.
1020.           //                              1. C LCUL R(x) i xP
1021.
1022.           //              1.1 C LCUL R(u) i uP
1023.
1024.           for(int j=1;j<=(M);j++)
1025.           {
1026.                   for(int i=1;i<=(N-1);i++)
1027.                   {
1028.                           Ae=fabs(Yc[i][j]-Yc[i][j-1]); //Ae=Aw
1029.                           An=fabs(X[i+1][j]-X[i][j]); //An=As
1030.
1031.                           //              Viscositats
1032.                           mu_w=mu[i][j];
1033.
1034.                           mu_e=mu[i+1][j];
1035.
1036.                           pos_x=Xu[i][j];
1037.                           pos_y=Yc[i][j];
1038.                           mu n=valor punt(pos x, pos y, mu, X, Y);
1039.
1040.                           pos_x=Xu[i][j];
1041.                           pos_y=Yc[i][j-1];
1042.                           mu_s=valor_punt(pos_x, pos_y, mu, X, Y);
1043.
1044.                           //              Fluxos m ssics
1045.                           m_punt_n=rho*v[i][j]*(Xu[i][j]-
   X[i][j])+rho*v[i+1][j]*(X[i+1][j]-Xu[i][j]);
1046.                           m_punt_s=rho*v[i][j-1]*(Xu[i][j]-
   X[i][j])+rho*v[i+1][j-1]*(X[i+1][j]-Xu[i][j]);
```

```
1047.
1048.                               m_punt_w=Ae*rho*(u[i-1][j]+u[i][j])/2;
1049.                               m_punt_e=Ae*rho*(u[i][j]+u[i+1][j])/2;
1050.
1051.
1052.                               //              Velocitats a les cares
1053.                               ue=esquema_upwind(u[i][j], u[i+1][j],
   m_punt_e);
1054.                               uw=esquema_upwind(u[i-1][j], u[i][j],
   m_punt_w);
1055.                               un=esquema_upwind(u[i][j], u[i][j+1],
   m_punt_n);
1056.                               us=esquema_upwind(u[i][j-1], u[i][j],
   m_punt_s);
1057.
1058.                               if(i>=2 && i<=(N-2) && j>=2 && j<=(M-1))
1059.                               {
1060.                                       ue=esquema_conveccio_difusio(X[i+1][j
   ], u[i-1][j], Xu[i-1][j], u[i][j], Xu[i][j], u[i+1][j], Xu[i+1][j],
   u[i+2][j], Xu[i+2][j], m_punt_e);
1061.                                       uw=esquema_conveccio_difusio(X[i][j],
   u[i-2][j], Xu[i-2][j], u[i-1][j], Xu[i-1][j], u[i][j], Xu[i][j], u[i+1][j],
   Xu[i+1][j], m_punt_w);
1062.                                       un=esquema_conveccio_difusio(Yc[i][j]
   , u[i][j-1], Yu[i][j-1], u[i][j], Yu[i][j], u[i][j+1], Yu[i][j+1],
   u[i][j+2], Yu[i][j+2], m_punt_n);
1063.                                       us=esquema_conveccio_difusio(Yc[i][j-
   1], u[i][j-2], Yu[i][j-2], u[i][j-1], Yu[i][j-1], u[i][j], Yu[i][j],
   u[i][j+1], Yu[i][j+1], m_punt_s);
1064.                               }
1065.
1066.
1067.                               //              R(u)
1068.
1069.                               //C?lcul dist?ncies entre nodes
1070.                               dPE=fabs(Xu[i+1][j]-Xu[i][j]);
1071.                               dPW=fabs(Xu[i][j]-Xu[i-1][j]);
1072.                               dPN=fabs(Yu[i][j+1]-Yu[i][j]);
1073.                               dPS=fabs(Yu[i][j]-Yu[i][j-1]);
1074.
1075.                               Ru[i][j]=(-(m_punt_e*ue-
   m_punt_w*uw+m_punt_n*un-m_punt_s*us)+(mu_e*Ae*(u[i+1][j]-u[i][j])/dPE-
   mu_w*Ae*(u[i][j]-u[i-1][j])/dPW+mu_n*An*(u[i][j+1]-u[i][j])/dPN-
   mu_s*An*(u[i][j]-u[i][j-1])/dPS))/omegax[i][j];
1076.
1077.                               //              uP
1078.                               if(t==0) Ru0[i][j]=Ru[i][j];
1079.                               uP[i][j]=u[i][j]+(deltat/rho)*((3/2)*Ru[i][j]
   -(1/2)*Ru0[i][j]);
1080.                       }
1081.               }
1082.
1083.
1084.
1085.               //              1.2 C?LCUL R(v) i vP
1086.
1087.               for(int j=1;j<=(M-1);j++)
1088.               {
1089.                       for(int i=1;i<=(N);i++)
1090.                       {
1091.                               Ae=fabs(Y[i][j+1]-Y[i][j]); //Ae=Aw
```

```
1092.                               An=fabs(Xc[i][j]-Xc[i-1][j]); //An=As
1093.
1094.                               //              Viscositats
1095.                               mu_n=mu[i][j+1];
1096.
1097.                               mu_s=mu[i][j];
1098.
1099.                               pos_x=Xc[i-1][j];
1100.                               pos_y=Yv[i][j];
1101.                               mu_w=valor_punt(pos_x, pos_y, mu, X, Y);
1102.
1103.                               pos_x=Xc[i][j];
1104.                               pos_y=Yv[i][j];
1105.                               mu_e=valor_punt(pos_x, pos_y, mu, X, Y);
1106.
1107.
1108.                               //              Fluxos m□ssics
1109.                               m_punt_w=rho*u[i-1][j]*(Yv[i][j]-
   Y[i][j])+rho*u[i-1][j+1]*(Y[i][j+1]-Yv[i][j]);
1110.                               m_punt_e=rho*u[i][j]*(Yv[i][j]-
   Y[i][j])+rho*u[i][j+1]*(Y[i][j+1]-Yv[i][j]);
1111.
1112.                               m_punt_s=(Xc[i][j]-Xc[i-1][j])*rho*(v[i][j-
   1]+v[i][j])/2;
1113.                               m_punt_n=(Xc[i][j]-Xc[i-
   1][j])*rho*(v[i][j]+v[i][j+1])/2;
1114.
1115.                               //              Velocitats a les cares
1116.                               ve=esquema_upwind(v[i][j], v[i+1][j],
   m_punt_e);
1117.                               vw=esquema_upwind(v[i-1][j], v[i][j],
   m_punt_w);
1118.                               vn=esquema_upwind(v[i][j], v[i][j+1],
   m_punt_n);
1119.                               vs=esquema_upwind(v[i][j-1], v[i][j],
   m_punt_s);
1120.
1121.
1122.                               if(i>=2 && i<=(N-1) && j>=2 && j<=(M-2))
1123.                               {
1124.                                       ve=esquema_conveccio_difusio(Xc[i][j]
   , v[i-1][j], Xv[i-1][j], v[i][j], Xv[i][j], v[i+1][j], Xv[i+1][j],
   v[i+2][j], Xv[i+2][j], m_punt_e);
1125.                                       vw=esquema_conveccio_difusio(Xc[i-
   1][j], v[i-2][j], Xv[i-2][j], v[i-1][j], Xv[i-1][j], v[i][j], Xv[i][j],
   v[i+1][j], Xv[i+1][j], m_punt_w);
1126.                                       vn=esquema_conveccio_difusio(Y[i][j+1
   ], v[i][j-1], Yv[i][j-1], v[i][j], Yv[i][j], v[i][j+1], Yv[i][j+1],
   v[i][j+2], Yv[i][j+2], m_punt_n);
1127.                                       vs=esquema_conveccio_difusio(Y[i][j],
   v[i][j-2], Yv[i][j-2], v[i][j-1], Yv[i][j-1], v[i][j], Yv[i][j], v[i][j+1],
   Yv[i][j+1], m_punt_s);
1128.                               }
1129.
1130.                               //              R(v)
1131.
1132.                               //C□lcul dist□ncies entre nodes
1133.                               dPE=fabs(Xv[i+1][j]-Xv[i][j]);
1134.                               dPW=fabs(Xv[i][j]-Xv[i-1][j]);
1135.                               dPN=fabs(Yv[i][j+1]-Yv[i][j]);
1136.                               dPS=fabs(Yv[i][j]-Yv[i][j-1]);
```

```
1137.
1138.                           Rv[i][j]=(-(m_punt_e*ve-
   m_punt_w*vw+m_punt_n*vn-m_punt_s*vs)+(mu_e*Ae*(v[i+1][j]-v[i][j])/dPE-
   mu_w*Ae*(v[i][j]-v[i-1][j])/dPW+mu_n*An*(v[i][j+1]-v[i][j])/dPN-
   mu_s*An*(v[i][j]-v[i][j-1])/dPS))/omegay[i][j];
1139.
1140.                           //              vP
1141.                           if(t==0) Rv0[i][j]=Rv[i][j];
1142.                           vP[i][j]=v[i][j]+(deltat/rho)*((3/2)*Rv[i][j]
   -(1/2)*Rv0[i][j]);
1143.                   }
1144.           }
1145.
1146.
1147.
1148.
1149.           //                              2. RESOLUCI CAMP PRESSIONS
   (EQUACI DE POISSON)
1150.
1151.
1152.           //      2.1 Introducci de les condicions de contorn als
   coeficients de pressi
1153.           coeficients_pressio_cc(N, M, ap, ae, aw, an, as, bp);
1154.
1155.           //mostrar_coeficients(N, M, aw, ae, as, an, bp, ap);
1156.
1157.
1158.           //      2.2 C lcul dels coeficients de pressi
1159.           for(int j=1;j<=M;j++)
1160.           {
1161.                   for (int i=1;i<=N;i++)
1162.                   {
1163.                           //C lcul rees
1164.                           Ae=fabs(Yc[i][j]-Yc[i][j-1]); //Ae=Aw
1165.                           An=fabs(Xc[i][j]-Xc[i-1][j]); //An=As
1166.
1167.                           //C lcul dist ncies entre nodes
1168.                           dPE=fabs(X[i+1][j]-X[i][j]);
1169.                           dPW=fabs(X[i][j]-X[i-1][j]);
1170.                           dPN=fabs(Y[i][j+1]-Y[i][j]);
1171.                           dPS=fabs(Y[i][j]-Y[i][j-1]);
1172.
1173.                           //C lcul coeficients
1174.                           ae[i][j]=Ae/dPE;        aw[i][j]=Ae/dPW;
     an[i][j]=An/dPN;        as[i][j]=An/dPS;
1175.                                                   ap[i][j]=ae[i][j]+aw[i][j]+an
   [i][j]+as[i][j];
1176.                           bp[i][j]=-(rho*uP[i][j]*Ae-rho*uP[i-
   1][j]*Ae+rho*vP[i][j]*An-rho*vP[i][j-1]*An)/deltat;
1177.                   }
1178.           }
1179.
1180.
1181.
1182.           //mostrar_coeficients(N, M, aw, ae, as, an, bp, ap);
1183.
1184.           //      2.3 Valors suposats
1185.
1186.           if(t==0)
1187.           {
```

```
1188.                       for(int j=0;j<=(M+1);j++)
1189.                       {
1190.                               for (int i=0;i<=(N+1);i++)
1191.                               {
1192.                                       p[i][j]=1;
1193.                               }
1194.                       }
1195.               }
1196.
1197.
1198.               //      2.4 Resoluci\xf4 del sistema (solver)
1199.               //Gauss_Seidel(N, M, e, fr, p, ap, ae, aw, an, as, bp);
1200.               Line_by_line_solver(N, M, e, p, aw, ae, as, an, bp, ap);
1201.
1202.
1203.
1204.
1205.
1206.               //                                  3. C\xf4LCUL CAMP DE VELOCITATS
1207.
1208.
1209.               for(int j=1;j<=(M);j++)
1210.               {
1211.                       for(int i=1;i<=(N-1);i++)
1212.                       {
1213.                               u[i][j]=uP[i][j]-(deltat/rho)*(p[i+1][j]-
   p[i][j])/(X[i+1][j]-X[i][j]);
1214.                       }
1215.               }
1216.
1217.
1218.               for(int j=1;j<=(M-1);j++)
1219.               {
1220.                       for(int i=1;i<=(N);i++)
1221.                       {
1222.                               v[i][j]=vP[i][j]-(deltat/rho)*(p[i][j+1]-
   p[i][j])/(Y[i][j+1]-Y[i][j]);
1223.                       }
1224.               }
1225.
1226.
1227.               //      C\xf4lcul variaci\xf4 de les velocitats en el temps
1228.
1229.               dif_u=0, dif_v=0; //Cal reinicialitzar el valor!
1230.
1231.               for(int j=1;j<=(M);j++)
1232.               {
1233.                       for(int i=1;i<=(N-1);i++)
1234.                       {
1235.                               maxdif(u[i][j], u0[i][j], dif_u);
1236.                       }
1237.               }
1238.
1239.               for(int j=1;j<=(M-1);j++)
1240.               {
1241.                       for(int i=1;i<=(N);i++)
1242.                       {
1243.                               maxdif(v[i][j], v0[i][j], dif_v);
1244.                       }
1245.               }
1246.
```

```
1247.
1248.           dif=max(dif_u,dif_v);
1249.           dif=dif/deltat;
1250.
1251.
1252.           //      Assignacions del pas temporal
1253.
1254.
1255.           for(int j=1;j<=(M);j++)
1256.           {
1257.                   for(int i=1;i<=(N-1);i++)
1258.                   {
1259.                           Ru0[i][j]=Ru[i][j];
1260.                           u0[i][j]=u[i][j];
1261.                   }
1262.           }
1263.
1264.           for(int j=1;j<=(M-1);j++)
1265.           {
1266.                   for(int i=1;i<=(N);i++)
1267.                   {
1268.                           Rv0[i][j]=Rv[i][j];
1269.                           v0[i][j]=v[i][j];
1270.                   }
1271.           }
1272.
1273.           cout<<"dif="<<dif<<" deltat="<<deltat<<" t="<<t<<endl;
1274.
1275.
1276.
1277.           t=t+deltat;
1278.
1279.
1280.
1281.   }//fi 'while' estat transitori
1282.
1283.           cout<<"Estat estacionari en t = "<<t<<" s"<<endl;
1284.
1285.           //              5. IMPRESSI? RESULTATS
1286.
1287.
1288.
1289.           ofstream myfile17;
1290.           myfile17.open ("velocitat_u.txt");
1291.
1292.           for(int i=0;i<=(N);i++)
1293.           {
1294.                   for(int j=(M+1);j>=0;j--)
1295.                   {
1296.                           myfile17<<u[i][j]<<" ";
1297.                   }
1298.
1299.                   myfile17<<endl;
1300.           }
1301.
1302.      myfile17.close();
1303.
1304.      ofstream myfile18;
1305.           myfile18.open ("velocitat_v.txt");
1306.
1307.           for(int i=0;i<=(N+1);i++)
```

```
1308.             {
1309.                     for(int j=(M);j>=0;j--)
1310.                     {
1311.                             myfile18<<v[i][j]<<" ";
1312.                     }
1313.
1314.                     myfile18<<endl;
1315.             }
1316.
1317.       myfile18.close();
1318.
1319.       ofstream myfile19;
1320.             myfile19.open ("pressio.txt");
1321.
1322.             for(int i=0;i<=(N+1);i++)
1323.             {
1324.                     for(int j=(M+1);j>=0;j--)
1325.                     {
1326.                             myfile19<<p[i][j]<<" ";
1327.                     }
1328.
1329.                     myfile19<<endl;
1330.             }
1331.
1332.       myfile19.close();
1333.
1334.
1335.       // Velocitat als nodes principals
1336.       vector <vector<double> > vel_u(N+2,vector<double> (M+2)),
   vel_v(vel_u); //Camp de pressions
1337.
1338.       velocitat_ppal(N, M, u_ref, vel_u, vel_v, u, v);
1339.
1340.       ofstream myfile20;
1341.             myfile20.open ("vel_u.txt");
1342.
1343.             for(int i=0;i<=(N+1);i++)
1344.             {
1345.                     for(int j=(M+1);j>=0;j--)
1346.                     {
1347.                             myfile20<<vel_u[i][j]<<" ";
1348.                     }
1349.
1350.                     myfile20<<endl;
1351.             }
1352.
1353.       myfile20.close();
1354.
1355.       ofstream myfile21;
1356.             myfile21.open ("vel_v.txt");
1357.
1358.       for(int i=0;i<=(N+1);i++)
1359.             {
1360.                     for(int j=(M+1);j>=0;j--)
1361.                     {
1362.                             myfile21<<vel_v[i][j]<<" ";
1363.                     }
1364.
1365.                     myfile21<<endl;
1366.             }
1367.
```

```
1368.       myfile21.close();
1369.
1370.
1371.       //                  IMPRESSI? VALORS COMPARATIUS
1372.
1373.       double y_res[17]={1, 0.9766, 0.9688, 0.9609, 0.9531, 0.8516, 0.734
   4, 0.6172, 0.5, 0.4531, 0.2813, 0.1719, 0.1016, 0.0703, 0.0625, 0.0547, 0};
1374.       double x_res[17]={1, 0.9688, 0.9609, 0.9531, 0.9453, 0.9063, 0.859
   4, 0.8047, 0.5, 0.2344, 0.2266, 0.1563, 0.0938, 0.0781, 0.0703, 0.0625, 0};
1375.       double val;
1376.
1377.       ofstream myfile22;
1378.            myfile22.open ("u_val_sim.txt");
1379.
1380.       for(int i=0;i<=16;i++)
1381.            {
1382.                    val=valor_punt(0.5, y_res[i], u, Xu, Yu);
1383.                    myfile22<<val<<" ";
1384.            }
1385.
1386.       myfile22.close();
1387.
1388.
1389.       ofstream myfile23;
1390.            myfile23.open ("v_val_sim.txt");
1391.
1392.       for(int i=0;i<=16;i++)
1393.            {
1394.                    val=valor_punt(x_res[i], 0.5, v, Xv, Yv);
1395.                    myfile23<<val<<" ";
1396.            }
1397.
1398.       myfile23.close();
1399.
1400.
1401.
1402.       vector <vector<double> > f_corrent(N+1,vector<double> (M+1));
1403.       funcions_corrent_u(f_corrent, N, M, Yc, u);
1404.
1405.
1406.       ofstream myfile24;
1407.            myfile24.open ("funcions_corrent.txt");
1408.
1409.       for(int i=0;i<=(N);i++)
1410.            {
1411.                    for(int j=(M);j>=0;j--)
1412.                    {
1413.                            myfile24<<f_corrent[i][j]<<" ";
1414.                    }
1415.
1416.                    myfile24<<endl;
1417.            }
1418.
1419.
1420.       myfile24.close();
1421.
1422.
1423.    }
```